Networks and Markets - Homework Assignment 2

Moises Baly

October 27, 2016

*Collaborated with Omri Sass (os98@cornell.edu), Claire Opila (co273@cornell.edu), Carolina Peisch (cgp48@cornell.edu)

Part 1

1

In the Travelers Dilemma we have that for every state, if players do not agree we subtract 2 from the player with higher price and add 2 to the player with lower price. This translates to our total utility decreasing over time. If we are in state $(5, 6)$ we will have utilities $u1 = 7$ and $u2 = 3$, for a total of 10, which will trigger a response on $p2$ to try to improve. This will be $(5, 4)$, with $u1 = 2$ and $u2 = 6$, for a total of 8. So we see that our utilities are decreasing. However, the definition of ordinal potential function is that it increases with every movement using BRD towards PNE. We can then simply define our ordinal potential function as $\Phi(a_1, a_2) = C_{max} - (u(a_1) + u(a_2))$, where C_{max} is the known maximum price in the game. For our travelers dilemma $C_{max} = 100$. As our total utilities decrease, our potential function will increase. However, for states $(2, 3)$ and $(3, 2)$, our total utilities will be 4, the same as $(2, 2)$, which would break our potential function. We need to somehow detect when both of the players choices are the same. We can do this by adding a small factor to our function that will cancel if both players choose the same price: $\Phi(a_1, a_2) = C_{max} - (u(a_1) + u(a_2)) - \frac{|u(a_1)-u(a_2)|}{u(a_1)+u(a_2)}$. Using this function will ensure that $(2, 2)$ will drop the adjustment at the end (fraction) but give slightly different values for non-equal choices.

2

(a) For BRD to converge, we've seen that we need an ordinal potential function: that is, a function that increases with every response from players trying to maximize their utility. In this case, we know that G_1 and G_2 have both ordinal potential functions, since they converge. This means that in G_1, move $u(a_i^1) < u(a_{i+1}^1)$ and in G_2, move $u(a_i^2) < u(a_{i+1}^2)$ using BRD. Since we know that in the new game G the utility of a play is the sum of utilities, we have that $u(a_{i+1}^G) = u(a_{i+1}^1) + u(a_{i+1}^2) > u(a_i^1) + u(a_i^2)$ if utilities are positive. This means we have an ordinal potential function in G and therefore BRD will converge. Since the games are independent and the players pick from the same set of actions, we can assume their behavior will be the same in G than in the other two

games.

(b) Yes it will. As stated above, G depends on the previous two games based on the utility function. If G reaches a PNE, it will mean that at least one of the other games has an ordinal potential function (utility grows over time when using BRD), and hence converges.

Part 2

Kleinberg and Eisley, exercise 19.8.1

(a) The nodes that will eventually switch to A will be c, i, k.
(b) The cluster blocking the spread is g, j, d, h, with density $2/3 > 1 - q = 1/2$.

Kleinberg and Eisley, exercise 19.8.2

(a) The first node to change is f, with a contagion factor of $\frac{2}{5}$. From there e follows, having 2 out of 4 connections switching. i is $\frac{2}{3}$ and therefore switches and at this point h's neighbors are all infected. g, j, k don't reach the necessary threshold.
(b) The cluster blocking the spread is g, j, k, with density $1 > 1 - q = 3/5$.
(c) Adding either g or j to the early adopters will result in a cascade. For example, adding g will cause j to have $\frac{2}{3}$ of its connections infected, causing it to switch. Then k will follow.

Kleinberg and Eisley, exercise 19.8.3

(a) The first node to change is k, with $\frac{1}{2}$ of its nodes already infected. Then follows l having 2 out of 3 connections infected. The rest do not meet the threshold.
(b) The cluster blocking the spread is h, c, d, i, n, m, with density 2/5 = 1 - q = 2/5.
**Note: we think there is a typo on the question, we can't find a cluster density of 3/5 in that graph.
(c) Adding any node in the blocking cluster h, c, d, i, n, m to the early adopters will result in a cascade. For example, adding d will cause i to switch, followed by j, n, m, h, g and c.

Kleinberg and Eisley, exercise 19.8.5

Let p be the proportion of neighbors that chose A and d the total number of neighbors. We want to compute the limit for which choosing A will yield more benefits than choosing B. We then have:

$pda + (1 - p)dx > (1 - p)db + pdx$.

Rearranging the terms we obtain that it is worth choosing A if $q \geq \frac{(b-x)}{(a+b-2x)}$

7

Lets consider the graph below with $q = 0.5$:

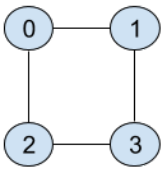

Figure 1

In this case, any two-nodes subset can be chosen to be our initial set of early adopters. The order in which we proceed will then change if the cascade will propagate to either A or B. Assume for example that $S = 0, 1$. If we then look at node 3, it will adopt A followed by 2, which will have no choice but follow suit. However, if we start by looking at 1, then its proportion of B neighbors will be 0.5 and it'll change to B, followed by 0. We can see that the cascade can go either way based on the order in which nodes start switching.

(b) The density of the set must be lower than the threshold value q in order to allow for the cascade to flow in both senses.

(c) The density of the set must be lower than $1 - q$ in order to allow for the cascade to flow in both senses (or the same as greater or equal than q).

Part 3

Kleinberg and Eisley, exercise 8.4.1

(a) If we send all cars through a single route, whether is UP or DOWN, we will have a travel time of 22 for any driver. Since the network is symmetric, we can

easily improve the utility for every driver by splitting the load in half. If we send 500 cars along every route, we will get a travel time of 17 for each driver. This cannot be improved: if we deviate from this behavior (i.e. 501 cars in one route vs 499 in the other), one of the two groups of drivers would worsen their utility. Therefore, the PNE is given by $x = 500$ and $y = 500$.

(b) $A \to C \to D \to B$ is an strictly dominant strategy and the only PNE. Assume that every driver has to decide whether to take $A \to C$ or $A \to D$. Even if every other driver was taking $A \to C$ (1000), the travel time (10) would still be better than $A - D$ (12). Therefore every driver will go the $A \to C$ route. Again, when deciding between $C \to D$ and $C \to B$, a later travel time of 10 in $D \to B$ (given $C \to D$ is 0) will be better than $C \to B$ (12). Hence every driver will take always $A \to C \to D \to B$. The equilibrium values of x and y will be both 1000. However, this PNE reduces the global utility for each driver to a travel time of 20, which is a worse than in the previous exercise, where we obtained a travel time of 17.

(c) The PNE will be achieved if half the cars go the upper route $A \to C \to B$ and the other half $A \to D \to B$. This happens because if half the cars go through $C \to D$ instead of $C \to B$, then the driving time would bump up to 15, due to having 1000 cars circulating through segment $D \to B$. Therefore, this would not constitute a PNE, leaving only the choice described before with $x = 500$ and $y = 500$. The cost of travel for each driver is 10, and 1000 drivers produces a total cost of travel of 10000. This would not change if road $C \to D$ was changed.

Kleinberg and Eisley, exercise 8.4.2

(a) Let $y = 80 - x$:

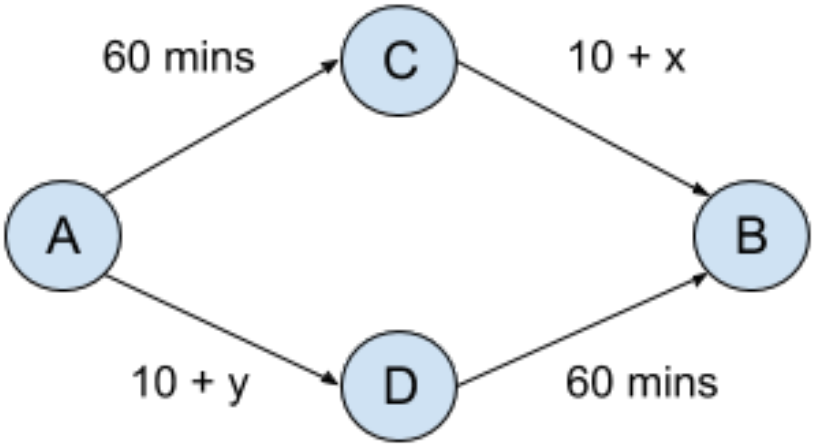

Figure 2: City travelers

(b) If we send all travelers through a single route, whether is I or II, we will have a travel time of 150 mins for any traveler. Since the network is symmetric, we can easily improve the utility for every driver by splitting the load in half. If we send 40 travelers along every route, we will get a travel time of 110 mins for each traveler. This cannot be improved: if we deviate from this behavior (i.e. 41 travelers in one route vs 39 in the other), one of the two groups of travelers would worsen their utility. Therefore, the PNE is given by $x = 40$.

(c) A PNE happens when the following scenario unfolds:
- 50 travelers take route $A \to D$.
- 30 travelers take route $A \to C \to B$.
- Out of the 50 travelers, 20 will try to improve by completing their route using $D \to C \to B$, while the remaining 30 will continue to B through D.

This gives us an equilibrium travel time of 120, which can not be improved.

(d) The previous total travel time was of $110 * 80 = 8800$, whereas our new travel time is $120 * 80 = 9600$

(e) If we send 30 travelers along the small routes, that is $A \to D \to C \to B$, we get a traveling time of 80 per traveler, with a total of $30 * 80 = 2400$ total travel time. If we send the remaining 50 travelers along the highways, that is $A \to C \to D \to B$, we obtain a total travel time of $120 * 50 = 6000$. The total travel time will then be $2400 + 6000 = 8400$, 400 minutes less than before the road was built. We can change the 30 travelers along the local streets in the range $(10, 40)$ non-inclusive and we will improve our initial solution without

extra roads.

Part 4

*Note: All code has dependencies to the *Graph*, *GraphOps*, *Path* and *Node* classes found in attached submission.

(10)

Nodes: Number of nodes in the graph. Number of persons.

Edges: Number of edges in the graph. Number of friendships.

Nodes in largest WCC: Number of nodes in the largest Weakly Connected Component. Since this is an undirected graph, the number means there is access from every node to every other in the graph (1 connected component). All of the persons in the graph are connected to each other in a directed or undirected way(through friends).

Edges in largest WCC: Number of edge in the largest Weakly Connected Component.

Nodes in largest SCC: Number of nodes in the largest Strongly Connected Component. Since the graph is undirected, SCC and WCC are the same.

Edges in largest SCC: Number of edge in the largest Strongly Connected Component.

Average clustering coefficient: The relation between the number of triplets of friends in which every friend is connected to each other and the number of triplets of friends in which two of them have a friend in common but are not connected to each other.

Number of triangles: Number of node triplets that form closed triangles. Groups of three friends in which each friend is connected to the other two (strong relationships).

Fraction of closed triangles: Actual number of closed triangles over the number of potential triangles

Diameter: Friendships have at most 8 degrees of separation.

90-percentile effective diameter: Diameter under which 90% of the nodes are connected. Description of graph cohesion, accounting for outliers that might skew the measure of the actual diameter.

11 - (a)

The code for the implementation of Ford-Fulkerson is as follows.

```
// Finds the max flow using Ford Fulkerson algorithm.
public static double maxFlow(Graph g, int source, int sink) {
 Graph residual = new Graph(g);
 double maxF = 0;
 double it = mfIteration(residual, source, sink);
 while (it != 1) {
  maxF += it;
  it = mfIteration(residual, source, sink);
 }
 return maxF;
}

private static double mfIteration(Graph g, int source, int sink) {
 Path p = bfs(g, source, sink);
 if (p != null && p.exists()) {
  ArrayList<Node> path = p.nodePath(g);
  double minC = Double.MAX_VALUE;
  //find min flow
  for (int i = 1; i < path.size(); i++) {
   if (path.get(i).c < minC)
     minC = path.get(i).c;
   }

   //update residual graph
   for (int i = 1; i < path.size(); i++) {
    Node n = path.get(i);
    n.c = minC;
    if (n.c <= 0) {
    // remove edge;
     g.removeEdge(path.get(i 1).ID, n.ID);
    }
    //add another edge or add to existing one.
    Node aux = g.findAdjacentNode(n.ID, path.get(i 1).ID);
    if (aux != null) aux.c += minC;
    else {
     g.connect(n.ID, path.get(i 1).ID, n.w, minC);
    }
  }
  return minC;
 }
 return 1;
}
```

mfIteration does an iteration of the Augmenting Paths algorithm. *maxFlow* is the high level function that keeps iterating until max flow is found.

We can verify our code on the following example graph:

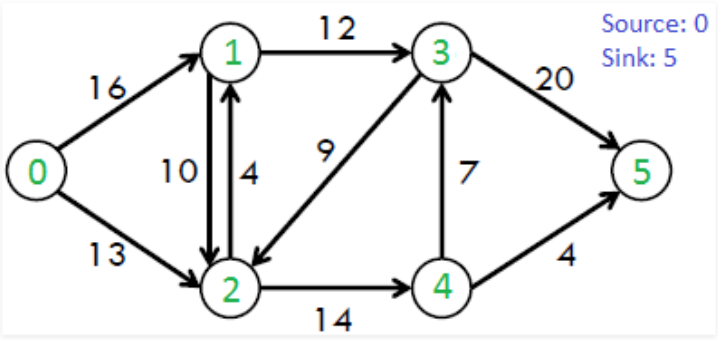

Figure 3: Graph 1

This graph has a Maximum Flow of 23. We run our code to verify:

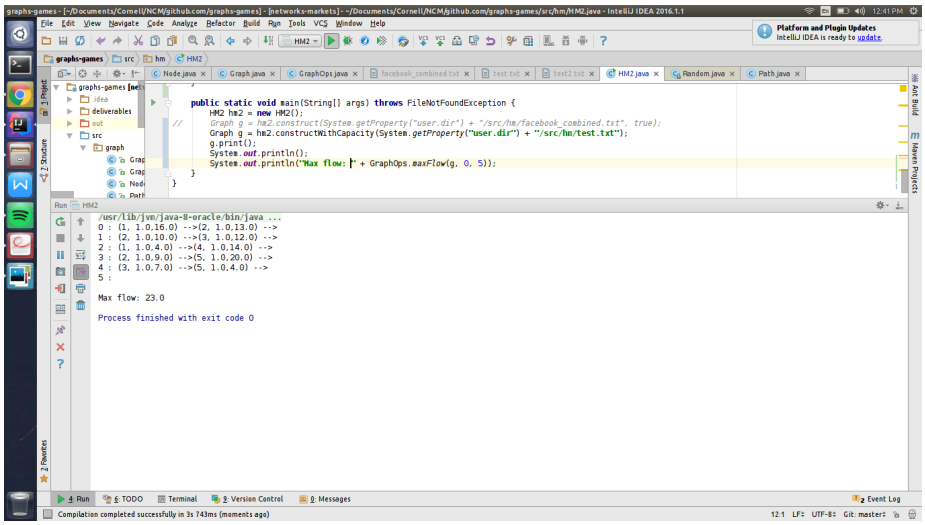

Figure 4: Running Maximum Flow on Graph 1

11 - (b)

From an purely implementation perspective, there is no difference between a directed or undirected graph. The Graph structure uses adjacency lists (see appendix), so an undirected edge (i, j) is actually $i - j$ AND $j - i$. With regards to capacity, a possible solution is to duplicate the capacity in both directions. See the next code that performs the undirected connection in our Graph (part of the *Graph* class in the appendix):

```
public void connectUnd(int i, int j, double w, double c) {
 ArrayList<Node> adj = adjacents(i);
 adj.add(new Node(j, w, c));
 nodes[i] = adj;
 adj = adjacents(j);
 adj.add(new Node(i, w, c));
```

```
 nodes[j] = adj;
}
```

In order to convert the Facebook graph from undirected to directed, all we have to do is use the above function to create "undirected" links. In the case of the Augmenting Paths algorithm, when traversing an edge reducing its flow by the minimum capacity, all we have to do is verify if there already an edge in the opposite sense, and if this is the case, add the min capacity to the existing edge capacity. The following snippet illustrates this operation (see algorithm in the previous section):

```
//add another edge or add to existing one.
Node aux = g.findAdjacentNode(n.ID, path.get(i    1).ID);
if (aux != null) aux.c += minC;
else {
 g.connect(n.ID, path.get(i    1).ID, n.w, minC);
}
```

In this way, our algorithm works for both directed and indirected graphs.

11 - (c)

We can use Ford-Fulkerson in a graph with capacities to 1 to find the number of disjoint paths. The code below illustrates the process:

```
public static double run10C(Graph g, int times) {
 double avg = 0;
 for (int k = 0; k < times; k++) {
  Random r = new Random(System.currentTimeMillis());
  int i = r.nextInt(g.size());
  int j = r.nextInt(g.size());
  double res = GraphOps.maxFlow(g, i, j);
  avg += res;
 }
 return avg / times;
}
```

We obtained an average of K-Disjoint Paths: 18.595.

11 - (d)

Intuitively, this means that in average, there are 18 different ways any given person is indirectly connected to another. This can also be viewed as the average elasticity of circles of friends. How many ways can we reach a person in another circle?

12 - (a)

The following code implements our contagion strategy. Please note that early adopter nodes are chosen randomly. We just specify how many we want, and

they are randomly generated.

```
public static HashSet<Integer> contagion(Graph g, int S, double q) {
 Random r = new Random(System.currentTimeMillis());
 HashSet<Integer> infected = new HashSet<>();
 for (int i = 0; i < S; i++) {
  int adopter = r.nextInt(g.size());
  infected.add(adopter);
 }
 while (true) {
  HashSet<Integer> result = contagion(g, infected, q);
  if (result.size() > infected.size()) {
   infected = result;
  } else {
   return infected;
  }
 }
}

public static HashSet<Integer> contagion(Graph g, HashSet<Integer> S, double q)
 HashSet<Integer> infected = new HashSet<>(S);
 boolean[] visited = new boolean[g.size()];
 Queue<Integer> queue = new LinkedList<>();

 for (int adopter : infected) {
  queue.add(adopter);
 }

 while (!queue.isEmpty()) {
  int x = queue.poll();
  visited[x] = true;

  //make infection decision
  int infectedCount = 0;
  for (Node n : g.adjacents(x)) {
   if (infected.contains(n.ID)) infectedCount++;
   if (!visited[n.ID]) queue.add(n.ID);
  }
  if (infectedCount / (double) (g.adjacents(x).size()) >= q) infected.add(x);
 }
 return infected;
}
```

We can then verify on the following graph:

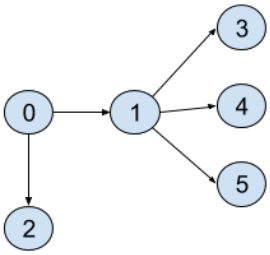

Figure 5: Graph 1

Running then different parameters will yield different levels of contagion:

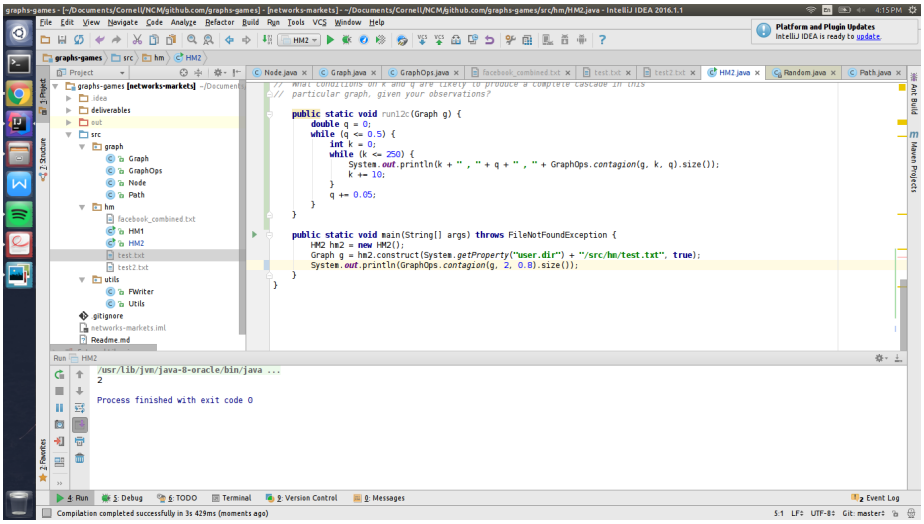

Figure 6: One early adopter and hard contagion environment

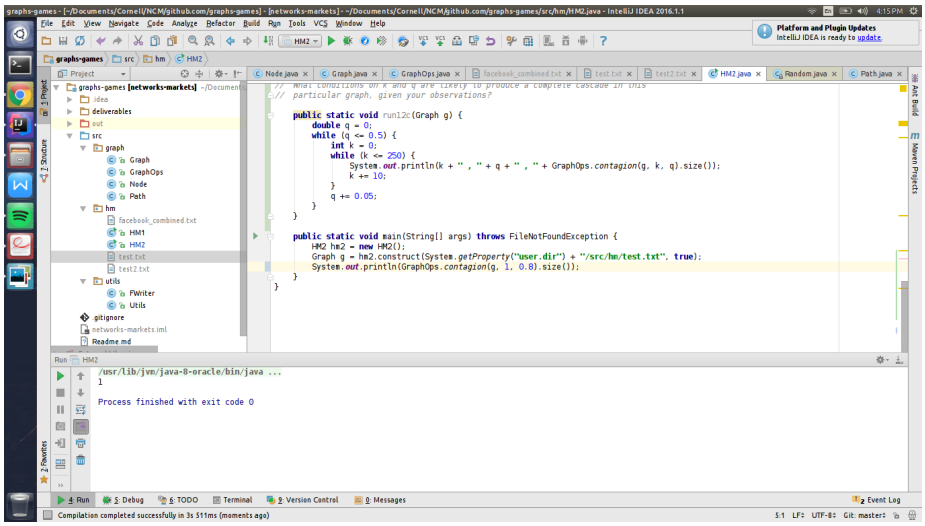

Figure 7: One early adopter and very easy contagion environment

12 - (b)

We can run as follows:

```
public static void run12b(Graph g) {
 int infected = 0;
 for (int i = 0; i < 100; i++) {
  infected += GraphOps.contagion(g, 10, 0.1).size();
 }
 System.out.println((double) infected / 100);
}
```

Achieving a result of 3283.27 infected nodes on average. Running the experiment multiple times shows that the average stays approximately between 3.2k and 3.3k.

12 - (c)

I ran this exercise as follows (a few more iterations than asked in the exercise):

```
public static void run12c(Graph g) {
 double q = 0;
 while (q <= 1) {
  int k = 0;
  while (k <= 500) {
   System.out.println(k + " , " + q + " , " + GraphOps.contagion(g, k, q).size(
   k += 10;
  }
  q += 0.05;
 }
}
```

And we can see how the contagion spreads or shrinks based on parameters:

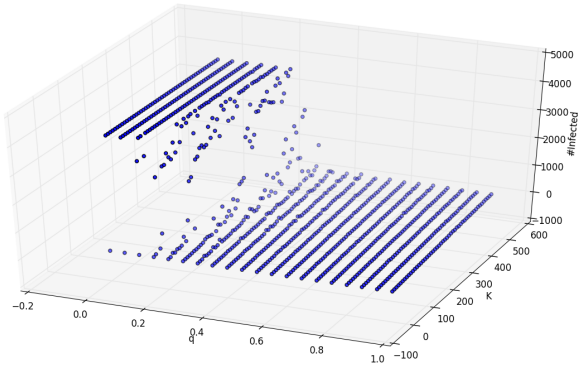

Figure 8: Trying different parameters - early adopters and contagion thresholds

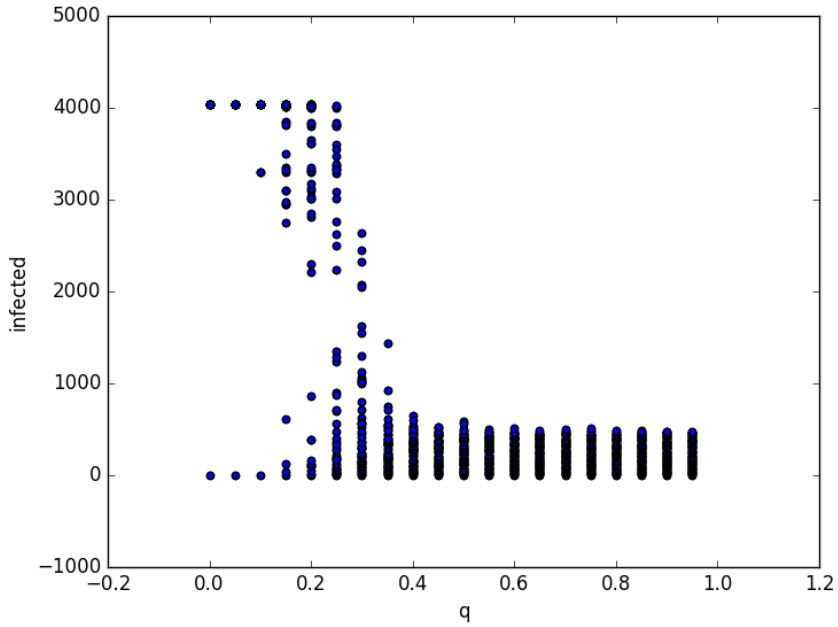

Figure 9: Trying different parameters - infected with respect to ‘q‘

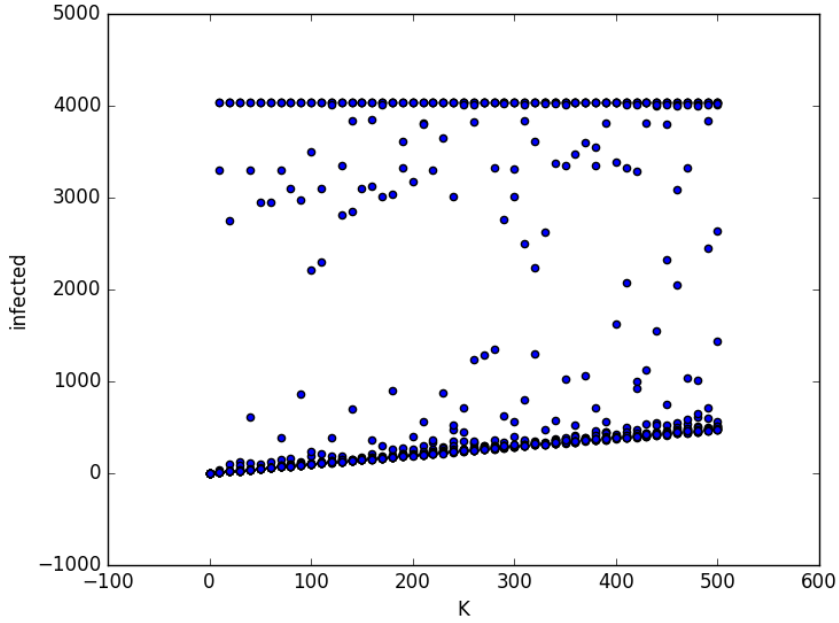

Figure 10: Trying different parameters - infected with respect to 'K'

Looking at the images above, it would appear as if q drives contagion in the range $[\ 0.1,\ 0.4]$, not fully making use of number of early adopters. Conversely, we can see that, when the threshold q becomes too hard for the disease to spread, there is a steady growth in the number of infected nodes as K increases - almost like brute force spread of the contagion. I guess the question would be what is the natural threshold in real settings and then how many early adopters do we need to include to produce a full cascade. Does such q exists uniformly for an entire network? For example, can determine with certainty how many friends would have to switch to Android for a given person to make the switch?

12 - (d)