Networks and Markets

Hw2 submission

Part 5: Experimental Evaluations

Alon Polski 206530461

Anna Petrenko 320460306

Ariel Chiskis 322442112

9.

An important comment:

It should take the code of question 9 around two and a half hours to run.

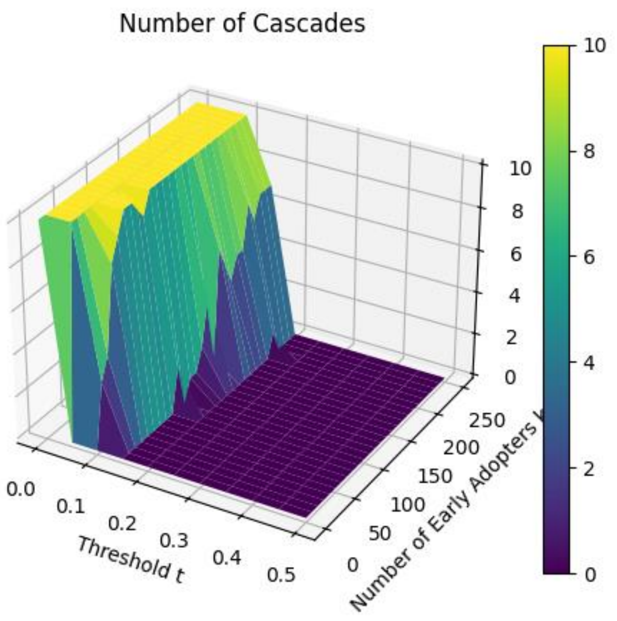
(b) We get a complete cascade in 28 of the BRD runs.

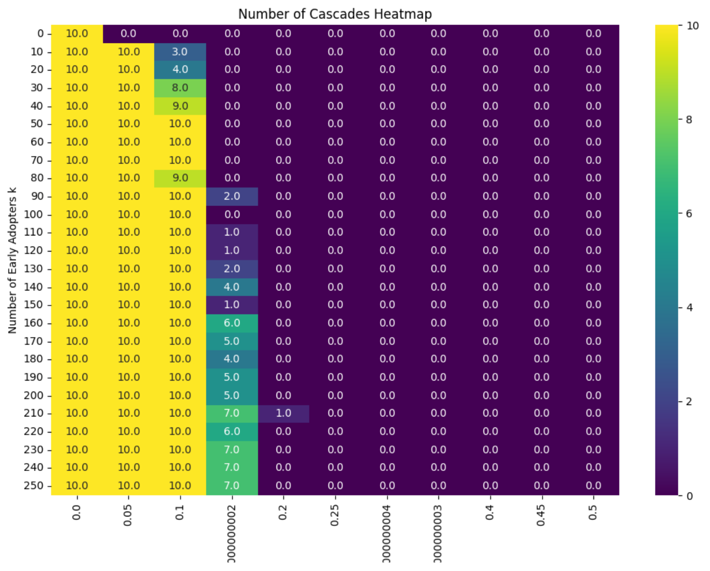
On average, there are 3438.23 nodes infected.

This means that there are some $1 - t = 0.9$ dense subsets of nodes in the graph and there's a non-neglectable probability for a node to be a member of such set, it also means there are not a lot of disjoint sets like this (less than 10 at lease.), because otherwise it would have been rare to infect all of them.

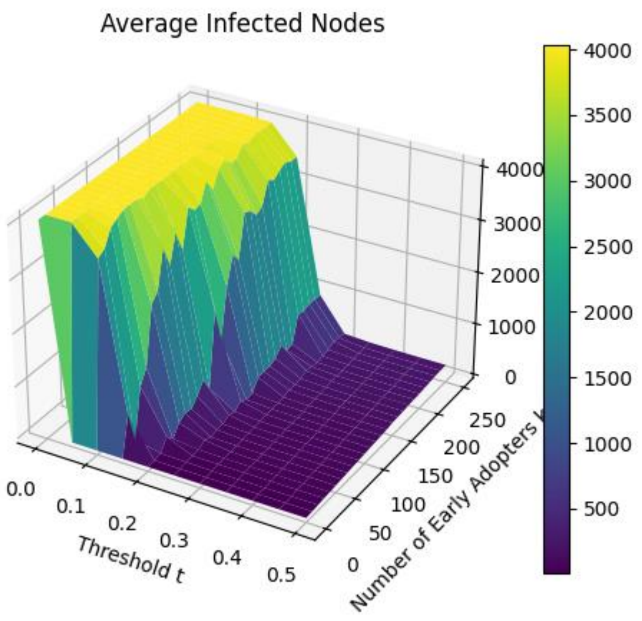
Also, we have a probability that is around $\frac{28}{100}$ to hit nodes in all of them if you sample 10 nodes randomly.

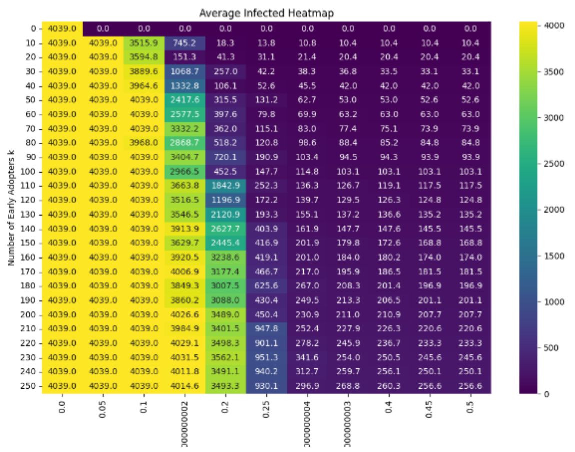
(c) The maximal value of t that seems to cause a complete cascade is around 0.2, but for such a high t the probability of a complete cascade is relatively low (around 0.1), we observe probability that is around 1 for a complete cascades for maximal t- around 0.1 where k is at least 50, and probability higher than $\frac{1}{2}$ for a complete cascade for maximal t- around 0.18, where k is at least 160. However small is k we can still see complete cascade if t is small enough but we start seeing cascades even for $k = 10$ where t is a most- around 0.1, for that k the probability for a complete cascade can get to around 1, when t is at most- around 0.05, for that k the probability of a complete can get to higher than $\frac{1}{2}$ where t is at most somewhere in $[0.05,1)$. As t decreases the probability for a complete cascade and the average size of a cascade seem to grow exponentially, as k increases the probability for a complete cascade as the average size of a cascade seem to grow linearly.





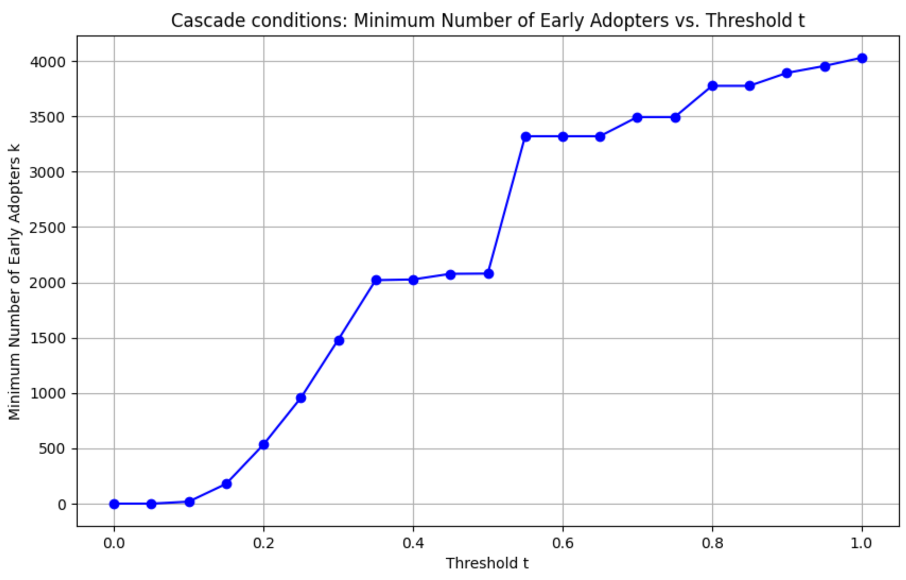
	0.0	0.05	0.1	000000002	0.2	0.25	000000004	000000003	0.4	0.45	0.5
0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	10.0	10.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20	10.0	10.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30	10.0	10.0	8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
40	10.0	10.0	9.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50	10.0	10.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
60	10.0	10.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70	10.0	10.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
80	10.0	10.0	9.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
90	10.0	10.0	10.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100	10.0	10.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
110	10.0	10.0	10.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
120	10.0	10.0	10.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
130	10.0	10.0	10.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
140	10.0	10.0	10.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
150	10.0	10.0	10.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
160	10.0	10.0	10.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
170	10.0	10.0	10.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
180	10.0	10.0	10.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
190	10.0	10.0	10.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
200	10.0	10.0	10.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
210	10.0	10.0	10.0	7.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
220	10.0	10.0	10.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
230	10.0	10.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
240	10.0	10.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
250	10.0	10.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0





	0.0	0.05	0.1	000000002	0.2	0.25	000000004	000000003	0.4	0.45	0.5
0	4039.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	4039.0	4039.0	3515.9	745.2	18.3	13.8	10.8	10.4	10.4	10.4	10.4
20	4039.0	4039.0	3594.8	151.3	41.3	31.1	21.4	20.4	20.4	20.4	20.4
30	4039.0	4039.0	3889.6	1068.7	257.0	42.2	38.3	36.8	33.5	33.1	33.1
40	4039.0	4039.0	3964.6	1332.8	106.1	52.6	45.5	42.0	42.0	42.0	42.0
50	4039.0	4039.0	4039.0	2417.6	315.5	131.2	62.7	53.0	53.0	52.6	52.6
60	4039.0	4039.0	4039.0	2577.5	397.6	79.8	69.9	63.2	63.0	63.0	63.0
70	4039.0	4039.0	4039.0	3332.2	362.0	115.1	83.0	77.4	75.1	73.9	73.9
80	4039.0	4039.0	3968.0	2868.7	518.2	120.8	98.6	88.4	85.2	84.8	84.8
90	4039.0	4039.0	4039.0	3404.7	720.1	190.9	103.4	94.5	94.3	93.9	93.9
100	4039.0	4039.0	4039.0	2966.5	452.5	147.7	114.8	103.1	103.1	103.1	103.1
110	4039.0	4039.0	4039.0	3663.8	1842.9	252.3	136.3	126.7	119.1	117.5	117.5
120	4039.0	4039.0	4039.0	3516.5	1196.9	172.2	139.7	129.5	126.3	124.8	124.8
130	4039.0	4039.0	4039.0	3546.5	2120.9	193.3	155.1	137.2	136.6	135.2	135.2
140	4039.0	4039.0	4039.0	3913.9	2627.7	403.9	161.9	147.7	147.6	145.5	145.5
150	4039.0	4039.0	4039.0	3629.7	2445.4	416.9	201.9	179.8	172.6	168.8	168.8
160	4039.0	4039.0	4039.0	3920.5	3238.6	419.1	201.0	184.0	180.2	174.0	174.0
170	4039.0	4039.0	4039.0	4006.9	3177.4	466.7	217.0	195.9	186.5	181.5	181.5
180	4039.0	4039.0	4039.0	3849.3	3007.5	625.6	267.0	208.3	201.4	196.9	196.9
190	4039.0	4039.0	4039.0	3860.2	3088.0	430.4	249.5	213.3	206.5	201.1	201.1
200	4039.0	4039.0	4039.0	4026.6	3489.0	450.4	230.9	211.0	210.9	207.7	207.7
210	4039.0	4039.0	4039.0	3984.9	3401.5	947.8	252.4	227.9	226.3	220.6	220.6
220	4039.0	4039.0	4039.0	4029.1	3498.3	901.1	278.2	245.9	236.7	233.3	233.3
230	4039.0	4039.0	4039.0	4031.5	3562.1	951.3	341.6	254.0	250.5	245.6	245.6
240	4039.0	4039.0	4039.0	4011.8	3491.1	940.2	312.7	259.7	256.1	250.1	250.1
250	4039.0	4039.0	4039.0	4014.6	3493.3	930.1	296.9	268.8	260.3	256.6	256.6

(d) We got the following graph:



We thought about adding on top of a binary search on k, a clustering algorithm to find dense disjoint sets, C_n, and then for each set find the node $v \in C_n$ that reaches the minimum of the function:

$$\frac{|C_n \cap N(v)|}{|N(v)|}$$

and then build S as the set of those nodes, but unfortunately, we did not have time to implement that and decided to just implement a binary search on k.

10.

(b)

i. Let

$$G \coloneqq (V, c)$$

be a capacitated graph, where:

$$V \coloneqq \{s, t\} \cup \{v_i\}_{i=1}^{n} \cup \{u_i\}_{i=1}^{m}$$

$$c(x, y) \coloneqq \begin{cases} 1, & (x = s) \land (y \in \{v_i\}_{i=1}^{n}) \\ 1, & (x \in \{u_i\}_{i=1}^{m}) \land (y = t) \\ 1, & (\exists i \in [n]. v_i = x) \land (\exists j \in [m]. u_j = y) \land (\mathit{driver\ i\ can\ pickup\ rider\ j}) \\ 0, & \mathit{else} \end{cases}$$

for every matching M, we can define the following flow in G:

$$f(x, y)$$

$$\coloneqq \begin{cases} 1, & (x = s) \land (\exists i \in [n]. v_i = y) \land (\mathit{driver\ i\ is\ matched}) \\ 1, & (\exists i \in [m]. u_i = x) \land (y = t) \land (\mathit{rider\ i\ is\ matched}) \\ 1, & (\exists i \in [n]. x = v_i) \land (\exists j \in [m]. y = u_j) \land (\mathit{driver\ i\ is\ matched\ to\ rider\ j}) \\ 0, & (x = s) \land \Big(\big((\exists i \in [n]. y = v_i) \land (\mathit{driver\ i\ is\ unmatched})\big) \lor (y \in \{u_i\}_{i=1}^{m}) \lor (y = t) \lor (y = s)\Big) \\ 0, & (\exists i \in [n]. v_i = x) \land \left(\big((\exists j \in [m]. u_j = y) \land (\mathit{driver\ i\ isn't\ matched\ to\ rider\ j})\big) \lor \left(y \in \{v_j\}_{j=1}^{n}\right) \lor (y = t)\right) \\ 0, & (\exists i \in [m]. x = u_i) \land \left(\left(y \in \{u_j\}_{j=1}^{m}\right) \lor \big((y = t) \land (\mathit{rider\ i\ unmatched})\big)\right) \\ 0, & x = y = t \\ -f(y, x), & \mathit{else} \end{cases}$$

And that's easy to see that $|f| = |M|$.

Also, let f' be an integer max flow in the graph, then we can define a matching M' by

$$(*)\ M' \coloneqq \{(i, j) : f(v_i, u_j) = 1\}$$

M is a matching because there's only one edge that goes into each v_i and its capacity is 1, and therefore, there can't be more than one u_j s.t $f(v_i, u_j) = 1$.

We've seen that for every matching M, there's a flow f, s.t. $|M| = |f|$, and that there's a matching M' for the integer max-flow f', s.t. $|f'| = |M|$, and therefore if there's an integer max-flow, its value is the maximal number of matches.

However, as a corollary from the Augmenting Path algorithm there's an integer max-flow, and therefore we can run on G the augmenting path algorithm to find an integer max-flow and output the maximal flow's value as the maximum number of matches.

ii. In part i, we've seen how to define a maximal matching from an integer max-flow in G (definition $(*)$), we can find that integer max-flow by running the augmenting path algorithm on G and then use definition $(*)$ to find the maximal matching.

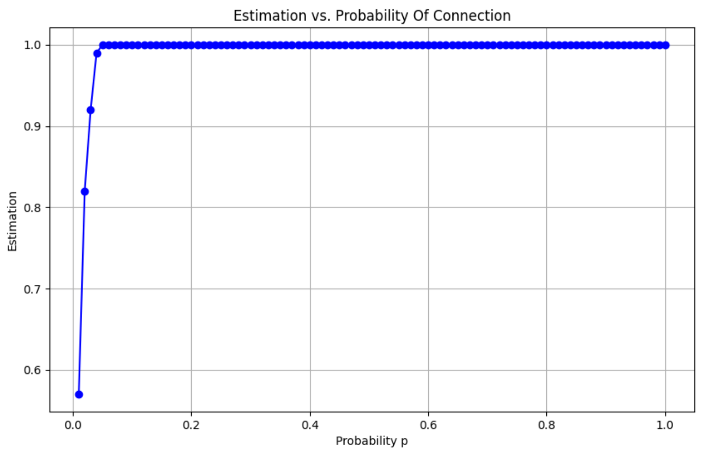
(c) In our first example, there were five drivers and five riders, driver 0 could only be matched to rider 1, driver 1 could only be matched to riders 0, 1, driver 2 could only be matched to rider 0, driver 3 could only be matched to riders 2 and 4, driver 4 could only be matched to riders 2, 3.

In that example we found out that the size of the maximal matching is four and an instance of such matching is the instance where driver 0 is matched to rider 1, driver 1 is matched to rider 0, driver 3 is matched to rider 2, rider 3 is matched to rider 3 and driver 3 and rider 4 are left unmatched.

In our second example, there were four drivers and four riders, where driver i could be matched to riders i and $i + 1$, except the last driver that could only be matched to rider 3.

In that example we found that the size of the maximal matching is 4 and that in that matching driver i is matched to rider i

(d) We fixed $n = 100$ because $n = 1000$ was too much. We got the following results:



to estimate the probability, we computed a case of n drivers and n riders where each driver could be connected to each rider with probability p, then we fixed the connections according to that p to 0 or 1, and divided the size of the maximal matching by p.