Homework Assignment 2 - Coding Part Write-up
Networks and Markets

Omer Zohar Gil Aharoni Adam Tuby

August 6, 2024

Part 5: Experimental Evaluations

1 Question 9

(a) It is evident that upon $t = 0.49$, the left example of Figure 4 would end up in a complete cascade, as $T \subseteq V \setminus S$ would simply be a set of either two connected nodes, or a single node, neither of which have a density of $1 - 0.49 = 0.51$. Conversely, upon $t = 0.51$, $T = V \setminus S$ has a density of $0.5 > 0.49 = 1 - 0.51$, and thus the cascade would never be complete per the theorem we've seen in class regarding cascades in the threshold model. We indeed see that our code always returns a complete cascade for $t = 0.49$ and never for $t = 0.51$.

For the right example of Figure 4, we can see that the cascade would be complete for $t = 0.32$, but not for $t = 0.34$, by similar means to the above. The maximum density of some set $T \subseteq V \setminus S$ is $\frac{2}{3}$, for $T' = V \setminus S$. Thus, for $t = 0.32$, for each $T \subseteq V \setminus S$, the density of T is less than $1 - t$, as $\frac{2}{3} = 1 - \frac{1}{3} < 1 - 0.32$, but for $t = 0.34$, the density of T' is greater than $1 - t$ as $\frac{2}{3} > 1 - 0.34$. Consequently, per the theorem we've seen in class, the cascade would be complete for $t = 0.32$ and not for $t = 0.34$. We indeed see that our code always returns a complete cascade for $t = 0.32$ and never for $t = 0.34$.

(b) Figure 1 Details the frequency histogram for the average amount of infected nodes (with X), for running our contagion BRD algorithm over small random sets ($k = 10$) with a low threshold ($t = 0.1$), for 100 times on the Facebook dataset.

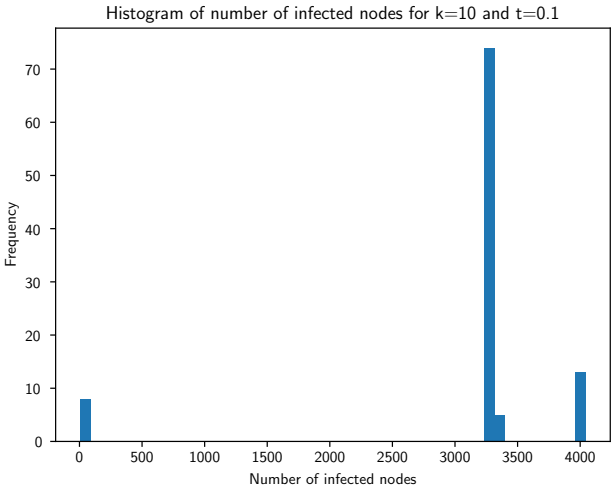


Figure 1: Frequency histogram for the average amount of infected nodes (with X), for running our contagion BRD algorithm over small random sets ($k = 10$) with a low threshold ($t = 0.1$), for 100 times on the Facebook dataset.

We observe 13/100 times that the graph has completely cascaded. Additionally, we observe an average of 3129.69 infected nodes. We observe that $\approx 10\%$ of runs had almost no infected nodes,

and $\approx 10\%$ of runs had a complete cascade (the 13/100 runs). The rest of the runs had a pretty fixed range for the amount of infected nodes - between 3200 and 3350 infected nodes.

(c) Figure 2 details the average rate of infected nodes (with X) for running our contagion BRD algorithm over varying k (increments of 10 from 0 to 250), with varying thresholds (increments of 0.05 from 0.05 to 0.5), for 10 times each on the Facebook dataset.

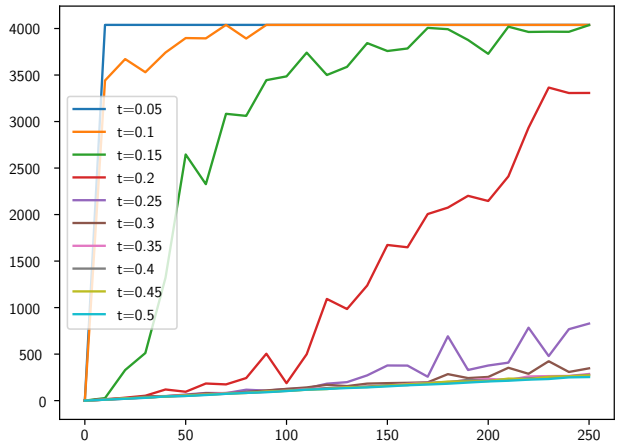


Figure 2: Average rate of infected nodes (with X) for running our contagion BRD algorithm over varying k (increments of 10 from 0 to 250), with varying thresholds (increments of 0.05 from 0.05 to 0.5), for 10 times each on the Facebook dataset.

It seems the graph does not cascade at all with $t > 0.15$, which means there are probably multiple sets with density of > 0.8.

It also seems that for every t there's a different k which might not adhere to some exact formula, but rather would depend on the number or mass of sets with density of > 0.8.
For example, for t=0.1 we found a cascading set first with $k = 10$, but for $t = 0.15$ it was with $k = 100$, and for $t = 0.2$, not even $k = 250$ sufficed for finding a cascading set.

Nevertheless, we observe a clear trend, that as k increases, and as t decreases, the average rate of infected nodes increases. This is intuitive, as the lower the threshold, the more likely a node is to be infected by its neighbors, and the larger the set of initial adopters, the better the chances an adjacent node to an initial adopter will surpass its threshold. As so, the lower t is, the lower k can be for a complete cascade to occur. Table 1 details the configuration and their conditions in order to consistently observe a complete cascade.

t	k's conditions
0.05	$k \geq 10$
0.1	$k \geq 90$
0.15	$k \geq 220$

Table 1: Configurations with high amount of complete cascades.

2 Bonus Question 2

For this exercise we implemented the search algorithm "simulated annealing". Since it is a known algorithm we won't expand on it here, though the implementation was heavily modified to fit our use-case. Our loss function was (number of non-infected) + (fraction size of S out of n). The nice thing about this loss function is that we always prioritize reducing the amount of non-infected at any cost for S, but when the number of non-infected is zero, we then focus on reducing the size of S. Another thing that we implemented is a weight function which gives each node a weight. For example, when S is not

cascading and we want to expand it, we might use a weight function based on each non-infected node's connections to other non-infected nodes. Instead of just taking the best node each time, we sample from the non-infected nodes using a probability distribution exponential in their weight. Another technique we employed is "doubling" - we start by adding a single node to S, then exponentially increase the number of nodes added (e.g., 1, 2, 4, 8, ...) until we hit a cascading set. This approach helps find an initial cascading set faster. Below is the resulting graph.

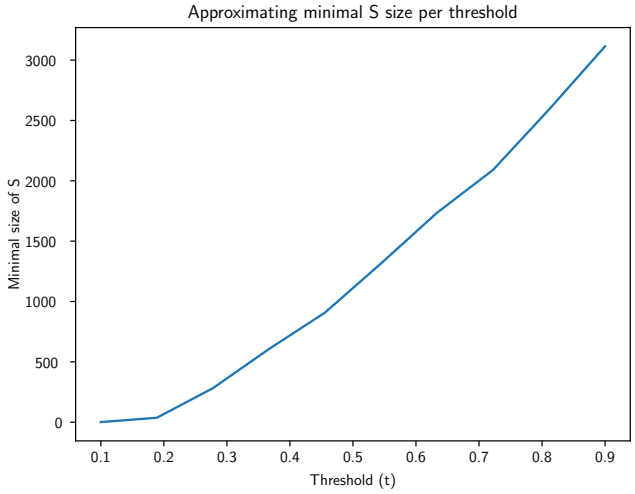


Figure 3: Finding minimal sized S per threshold

Note: We included bonus 2's code in the submission inside `hw2_p9.py`, and we added the analysis code of its usage in the main function. We implemented our algorithm using multiprocessing to speed up the process, and thus it sometimes arouses errors upon the first run. For that reason, we disabled the multiprocessing by default, but you can change it to use multiple processes in the `num_processes` keyword argument in the `run_optimizer_par` function—though beware, it might arouse errors upon the first run. We found that *if that happens, running it again* usually solves the issue.

3 Question 10

(b) Given a set of n drivers, m riders, and sets of possible riders that each driver can pick up:

(i) We can use the maximum-flow algorithm to determine the maximum *number* of matches that can be made. Given some $G = (V, E)$, where $V = D \cup R$, D is the set of drivers, R is the set of riders, and E is the set of edges, where $(d, r) \in E$ if and only if driver d can pick up rider r, we can construct a flow network $G' = (V', E', c)$, as illustrated in Figure 4, where $V' = V \cup \{s, t\}$—we add artificial sink and source nodes s and t respectively, such that $E' = E \cup \{(s, d) \mid d \in D\} \cup \{(r, t) \mid r \in R\}$, and $c \equiv 1$ (but 0 for non-existent edges). That is, we add an edge with capacity 1 from s to each driver, and from each rider to t. We then add an edge with capacity 1 from each driver to each rider in its list of compatible riders. The maximum flow value in network G' would be the maximum number of matches possible.

With similar arguments to the ones we've seen in class, this indeed gives us the maximum number of matches that can be made. We prove that

$$\max - flow = \max_{f \text{ is a flow in } G'} |f| = \max_{M \subseteq E \text{ is a matching}} |M| = max - matching$$

- **max − flow ≤ max − matching**: Let F be the maximum flow in G'. We can construct a matching M from F for G by taking the set of edges

$$M_F = \{(d, r) \in E \mid F(d, r) > 0\} \overset{(1)}{=} \{(d, r) \in E \mid F(d, r) = 1\}$$

where (1) is derived from the fact that $F(d, r) \in \mathbb{N}$ and as $F(d, r) \leq c(d, r) = 1$. Now, suppose two edges $(d, r), (d, r')$ in M_F share the same source node d. As

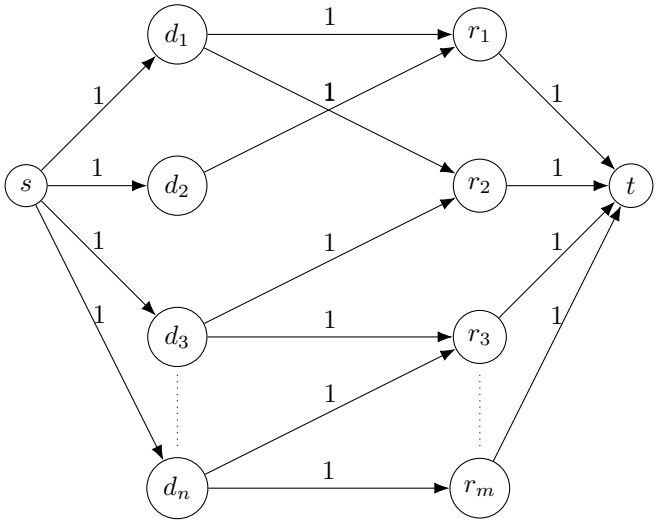


Figure 4: Flow network for the maximum number of matches that can be made. In this network, the list of compatible riders for d_1 is $\{r_1, r_2\}$, for d_2 is $\{r_1\}$, for d_3 is $\{r_2, r_3\}$, ... ,and for d_n is $\{r_3, r_m\}$.

$$\sum_{v \in V'} F(v, d) = F(s, d) \leq 1$$

and due to conservation of flow, we get that

$$1 \geq \sum_{v \in V'} F(v, d) = \sum_{v \in V'} F(d, v) \geq F(d, r) + F(d, r') = 2$$

which is a contradiction. Thus, no two edges in M_F share the same source node. Symmetrically, no two edges in M_F share the same target node. Thus, M_F is a matching. Moreover,

$$|F| = \sum_{v \in V'} F(s, v) = \sum_{d \in D} F(s, d) \overset{\text{conservation of flow}}{=} \sum_{d \in D} \sum_{r \in R} F(d, r) = \sum_{\substack{(d,r) \in D \times R \\ F(d,r) > 0}} F(d, r) = \sum_{(d,r) \in M_F} F(d, r) = \sum_{(d,r) \in M_F} 1 = |M_F|$$

Thus, $|M_F| = |F| = max - flow$, and therefore $max - flow \leq max - matching$.

- **max − matching ≤ max − flow**: Let M be some maximum matching in G. We can construct a flow F_M from M for G' as follows:

$$F_M(e) = \begin{cases} 1 & \text{if } e = (s, d) \text{ where } d \text{ is matched in } M \\ 1 & \text{if } e = (r, t) \text{ where } r \text{ is matched in } M \\ 1 & \text{if } e = (d, r) \text{ where } (d, r) \in M \\ 0 & \text{otherwise} \end{cases}$$

It is clear that $\forall v, u \in V', F_M(v, u) \leq c(v, u)$, additionally, as M is a matching, conversation of flow is maintained: let $d \in D$, then

$$\sum_{v \in V'} F_M(v, d) = F_M(s, d) = 1 = |\{r \in R \mid (d, r) \in M\}| = \sum_{v \in V'} F_M(d, v)$$

Thus F_M conserves flow for nodes in D, and symmetrically so for nodes in R. Thus, F_M is a flow. Moreover, as M is a matching and matches each node at most once, we get that

$$|F_M| = \sum_{v \in V'} F_M(s, v) = \sum_{d \in D} F_M(s, d) = \sum_{d \text{ is matched in } M} F_M(s, d) = \sum_{d \text{ is matched in } M} 1 = |\{d \in D \mid d \text{ is matched in } M\}| = |M|$$

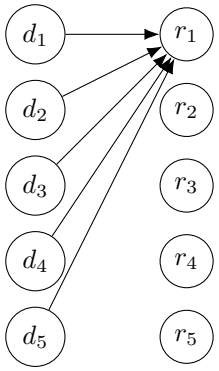
Therefore, $|F_M| = |M| = max - matching$, and thus $max - matching \leq max - flow$.

Thus, $max - flow = max - matching$, and the maximum flow algorithm can be used to determine the maximum number of matches that can be made.

(ii) To find the actual maximum number of matches, we can use the **Augmenting Path Algorithm** we've seen in class to find the maximum-flow in the network G', and then extract a matching from the flow which would have the same cardinality (and thus be the maximum matching per (i)). The extraction is detailed in the proof of $max - flow \leq max - matching$ above.

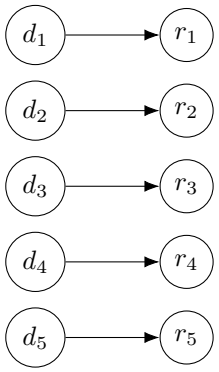
(c) The following are the examples we tested our maximal matching algorithm for Uber drivers and riders, which can be run through `max_matching_sanity_checks()` in our code. They all detail a bipartite graph between 5 drivers and 5 riders, and the compatibility between them represented as edges.

(1) All drivers can only pick the first rider:



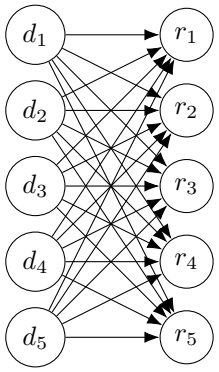
The maximum matching is clearly 1, as only one rider can be matched with a driver. Indeed, the maximal matching algorithm returns $[0, None, None, None, None]$—which in our code it stands for the first driver being matched with the first rider, and the rest of the drivers being unmatched.

(2) For every i, driver i can pick rider i:



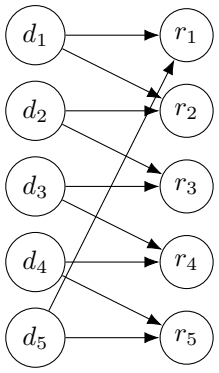
The maximum matching is clearly 5, as each driver can be matched with a rider. Indeed, the maximal matching algorithm returns $[0, 1, 2, 3, 4]$—which in our code it stands for each driver being matched with the corresponding rider.

(3) All drivers can pick all riders:



The maximum matching is clearly 5, as each driver can be matched with any rider, which in particular means that each driver can be matched with a different rider. Indeed, the maximal matching algorithm returns a list full of distinct non-None values, implying that each driver is matched with a rider, and that each driver is matched with a different rider.

(4) For every i, driver i can pick riders $i, (i + 1) \mod 5$:



The maximum matching is clearly 5, as we can observe easily that driver d_i can be matched with rider r_i such that each driver is matched with a different rider. Indeed, the maximal matching algorithm returns a list full of non-None distinct values, where each driver is matched with a rider in its list of compatible riders.

(d) We consider the case where there are n drivers and n riders, where each driver is connected to each rider with probability p. Fixing $n = 100$, we run the maximal matching algorithm for 100 times for each varying values of p—increments of 0.01 from 0.03 to 0.1, and use those measurements of the size of the found maximal matching, to estimate the probability that all n riders will get matched (by computing the sample mean of the measurements being a maximal matching of size n).

At first we tried estimating using ps from $\{0.1, 0.2, 0.3, 0.4, 0.5\}$, but we found out that even for $p = 0.1$ there's a very high probability of getting a full match.
That's when we started using $\{0.03, 0.04, 0.05, 0.06, 0.06, 0.07, 0.08, 0.09, 0.1\}$, where the differences were noticeable.

Figure 5 details the results of this experiment—as a plot of the estimated probability that all n riders will get matched as a function of p. We observe the intuitive trend of the probability increasing as p increases, as the higher the probability of a driver being connected to a rider, the more riders each driver will likely be connected to, and thus the more likely it is that all riders will get matched. We also observe that the probability is very low for $p = 0.03$, and increases rapidly as p increases, and then starts to plateau around $p = 0.08$.

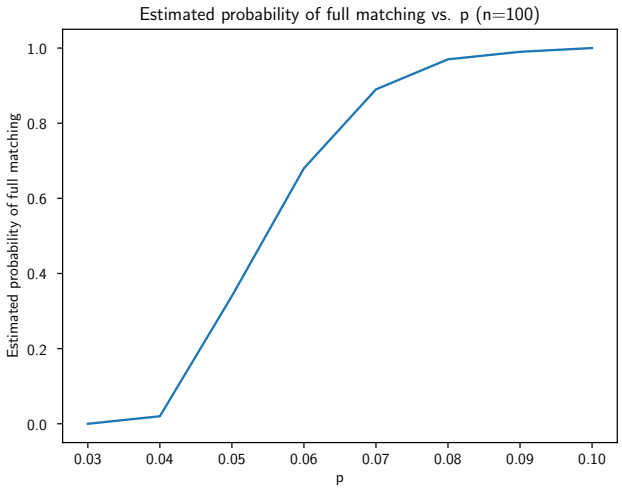


Figure 5: Estimated probability that all n riders will get matched as a function of p.

4 Bonus Question 3

4.1 Asymptotic Solution - a Sketch

From the paper of Erdős and Rényi [1], let $m(n) = n(\log n + c_n)$. Then $P[B_{n,m(n)}$ has a P.M.$] \to 1$ *if and only if* $c_n \to \infty$ when $n \to \infty$.

We know that given a probability $p(n)$, we have $E[m(n)] = n^2 p(n)$. This means that $P[m(n) > \frac{1}{2}n^2 p(n)] \to 1$. Actually, $P[m(n) > (1-\epsilon)n^2 p(n)] \to 1$ for every $\epsilon > 0$.

Now, we need to choose $p(n)$ such that $E[m(n)] = n(\log n + c_n)$ for a series c_n s.t. $c_n \to \infty$. We take $\frac{1}{2}n^2 p(n) = n(\log n + c_n)$.

Solving for $p(n)$: $p(n) = \frac{2}{n}(\log n + c_n)$

Note that we can use $c_n = \log n$, which approaches infinity, and we get: $p(n) = 4\frac{\log n}{n}$

We can even choose a slower-growing function for c_n, but it won't change much. As discussed, we can change $\frac{1}{2}$ to $1-\epsilon$, then we get:

$p(n) = \frac{2}{1-\epsilon}\frac{\log n}{n}$ for $\epsilon > 0$

A little more rigorously, we found that for $\epsilon > 0$, we get:
$P[m(n) = n(\log n + c_n)$ such that $c_n \to \infty] \to 1$

and thus: $P(P(B_{n,m(n)}$ has a P.M.$) \to 1) \to 1$

This way of writing things might look unusual but is very common in probabilistic analysis.

4.1.1 Asymptotic Solution's Case Study: $c_n = \sqrt{n}$

Considering the asymptotic solution of Section 4.1, we choose $c_n = \sqrt{n} \overset{n\to\infty}{\longrightarrow} \infty$, leading to $p(n) \approx \frac{1}{\sqrt{n}}$.

Figure 6 details a plot of the minimum p required for the probability of having a perfect matching to rise above 0.99, as a function of n, using binary search to find the minimum p for varying values of n (increments of 5 from 10 to 100). We observe fast convergence to the asymptotic bound, as expected.

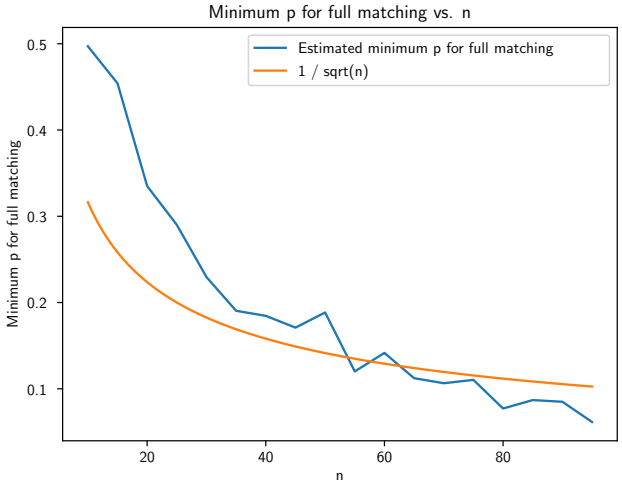


Figure 6: Minimum p required for the probability of having a perfect matching to rise above 0.99, as a function of n, for $c_n = \sqrt{n}$.

4.2 Numeric Solution - a Sketch

For every subset S let $N(S)$ be its set of neighbors. A set *fails* if $|S| > |N(S)|$. Let S be some set of size k let v be a potential neighbors, what is the probability that $v \notin N(S)$? This is exactly $(1-p)^k$. This means that $p(v \in N(S)) = 1 - (1-p)^k$. Consequently, this means size of $N(S)$ is a distributed as Bin(q, n) where q is the probability of each node. Now as we saw, since S *fails* if $N(S)$ is smaller than $N(S)$, the probability that that will happen is exactly $\sum_{i=0}^{k-1}(n \text{ choose } i)(1-(1-p)^k)^i)(1-p)^{k(n-i)}$. From union bound we have $\sum_{k=1}^{n}(n \text{ choose } k)\sum_{i=0}^{k-1}(n \text{ choose } i)(1-(1-p)^k)^i)(1-p)^{k(n-i)}$. For any given probability, in our case for 99 percent, for any n we can plug it in and approximate p numerically.

References

[1] Alan Frieze and Boris Pittel. Perfect matchings in random graphs with prescribed minimal degree. In *Mathematics and Computer Science III: Algorithms, Trees, Combinatorics and Probabilities*, pages 95–132. Springer, 2004.