# The Graph Recommendation Problem

Arda Antikacioglu, R. Ravi, Srinath Sridhar

April 14, 2013

## 1 Introduction

As the amount of data a service collects and indexes increases, the problem of surfacing relevant content arises and may companies face this challenge today. Netflix needs to recommend similar movies, Amazon and Bloomreach need to recommend similar products, Facebook needs to recommend pages or friends. This recommendation problem is hard to solve and often critical to the success of these services. For example, when a user signs up for the first time, Facebook needs to recommend them to as many people relevant as possible very quickly because if a user does not need reach a certain friend count, his the probability of retaining that user decreases rapidly. Similarly, Amazon may want to surface an undersold product on more commonly visited product pages, but the revenue gain will not be significant enough unless a certain number of pages recommend the said product. What makes this problem difficult is that you can only show so many recommendations in high value spots without overwhelming the user. So the number of recommendations a high traffic user or product can make is limited, and these need to be allocated efficiently. This motivates modeling the problem as a an optimization problem on graphs as follows.

We let $L$ and $R$ be sets of products or users. We model the relationship between these sets of products or users by a bipartite graph $G = (L, R, E)$ where there is an edge between between a $u \in L$ and $v \in R$ if they are related in some way. Our goal is to find a subgraph $H \subseteq G$ such that $\deg_H(u) \leq c$ for all $u \in L$ and the number of vertices $u \in L$ with $\deg_G(u) \geq a$ is maximized. We will call this problem the $(c, a)$-graph recommendation problem.

This problem can capture all the examples described above. In the case of Netflix $L$ is a set of movies and $R$ is a set of users. In the case of Facebook $L$ is a set of active users and $R$ is a set of new users who need friends. In Amazon and Bloomreach's case $L$ is a set of high traffic/revenue products and $R$ is a set of products whose exposure we seek to increase.

There are special cases of the graph recommendation problem that can be solved to optimality in polynomial time. In particular if $a = 1$ then the problem is simply a maximum cardinality $b$-matching problem. However, implementing the algorithms that solve $b$-matchings such as the blossom algorithm in general graphs is tricky as they involve a good deal of bookkeeping. Furthermore they take $O(v^{2.5})$ time at best and are memory-intensive. With web-scale sizes of the problems encountered in practice, any solution that requires superlinear time is unlikely to be practical. In this paper we develop several linear time algorithms that operate locally and therefore use very little memory, but still output solutions that are near optimal in the average-case and worst case.

## 2 Summary of Results

Our main contributions are the following:

1. We describe several general models such as the Hierarchical Tree Model and the Cartesian Product Model which describe how graph recommendation graphs arise probabilistically.

2. For each of these models, we describe a simple randomized linear time algorithm that solves the graph recommendation problem to near optimality in expectation.

3. We also show that a simple linear time greedy algorithm can approximate the problem to a constant factor in the worst case.

4. We show that the models described in the problem can describe real-life recommendation graphs and that the random sampling algorithms we describe find near optimal solutions to the graph recommendation problem.

# 3 Models

We describe four different models that can generate recommendation graphs. The first model we describe is called the *fixed degree model*, which is analyzed in Section 4.1. In this model, every $u \in R$ is related to a fixed number of uniformly randomly chosen adjecent vertices $v \in R$. This model is interesting not for its expressive power but because of its simplicity. It is easy to analyze and also motivates the analysis of richer models.
In Section 4.4, we generalize the fixed degree model to a model where the graph is fixed and the edges can weights. In Section 4.2, we analyze the *hierarchical tree model* which aims to model recommendation trees that arise from relationships in a product hierarchy. Finally, in Section 4.3 we study the *cartesian product model* which aims to model recommendation graphs which arise from combining the relevancy advice of several different algorithms.

# 4 Results

## 4.1 Fixed Degree Model

In this model, we assume that a bipartite graph $G = (L, R, E)$ is generated probabilistically by the following procedure. Each vertex $v \in L$, uniformly samples a set of $d$ neighbors from $R$. For convenience let $|L| = l$, $|R| = r$ and $k = l/r$. From $G$, we sample a subgraph $H$ where for each vertex $u \in L$ a set of $c$ neighbors are sampled uniformly from set of incident vertices. The following theorem derives a lower bound on the expected size $S$ of the number $v \in R$ such that $\deg_H(v) \geq 1$.

**Theorem 1.** *Suppose that $G = (L, R, E)$ and $H \subseteq$ is generated as above. Then*

$$E[S] \geq r(1 - \exp(-ck))$$

*where the expectation is over the random sampling of $G$ and $H$.*

*Proof.* For each $v \in R$ let $X_v$ be the indicator variable for the event that $\deg_H(v) \geq 1$. Note that since for each vertex $u \in L$, $H$ uniformly samples from a uniformly sampled set of neighbors, we can think $H$ as being generated by the same process that generated $G$, but with $d$ replaced with $c$. Now for a specific vertex $u \in R$, the probability that it has no incident edges is $\left(1 - \frac{1}{r}\right)^c$. Since the selection of neighbors for each vertex in $L$ is independent, it follows that that:

$$\Pr[X_v = 0] = \left(1 - \frac{1}{r}\right)^{cl} \leq \exp\left(-c \cdot \frac{l}{r}\right) = \exp(-ck)$$

Note that $S = \sum_{v \in R} X_v$. Applying linearity of expectation, we get

$$\mathrm{E}[S] = \sum_{v \in V} \mathrm{E}[X_v] \geq r(1 - \exp(-ck))$$

□

While this shows a lower bound in absolute terms, we must compare it to the best possible solution *OPT*. The follow theorem proves the approximation ratio to *OPT*.

**Theorem 2.** *The above sampling algorithm gives a* $1 - 1/e$ *factor approximation to the* $(c, 1)$*-graph recommendation problem in expectation*

*Proof.* The size of the optimal solution is bounded above by both the number of edges in the graph and the number of vertices in $R$. The former of these is $cl = ckr$ and the latter is $r$, which shows that $OPT \leq r \max(ck, 1)$. Therefore, by simple case analysis the approximation ratio in expectation is at least

$$\frac{1 - \exp(-ck)}{\min(ck, 1)} \geq 1 - \frac{1}{e}$$

□

However in reality, the approximation obtained by this sampling approach can be much better for certain values of $ck$. In particular, if $ck > 1$, then the approximation ratio is $1 - \exp(-ck)$, which approaches 1 as $ck \to \infty$. In particular, if $ck = 3$, then the solution will be at least 95% as good as the optimal solution even with our trivial bounds. Similarly, when $ck < 1$, the approximation ratio is $(1 - \exp(-ck))/ck$ which also approaches 1 as $ck \to 0$. In particular, if $ck = 0.1$ then the solution will be at 95% as good as the optimal solution. The case when $ck = 1$ therefore represents the worst case outcome for this model where we only guarantee 63% optimality. The graph below shows the approximation ratio as a function of $ck$.

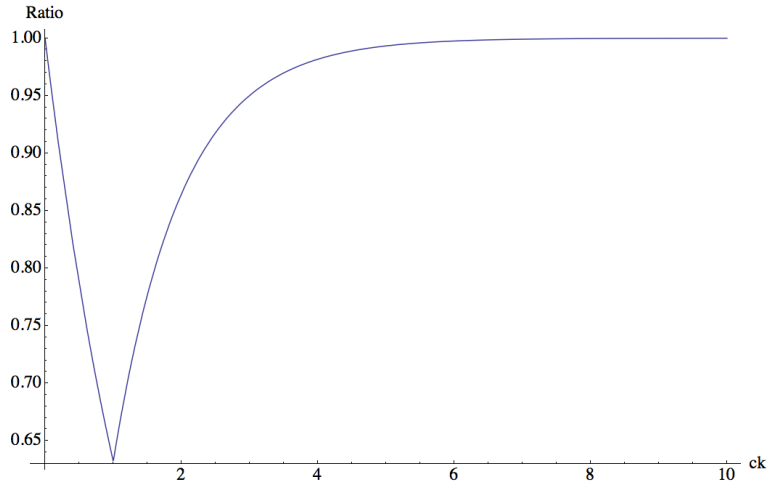


Figure 1: Approximation Ratio as a function of $ck$

Now suppose that $G$ is generated and $H$ is sampled using the same processes as described above. In the next theorem, we extend the above bounds to the $(c, a)$-graph recommendation problem where $a > 1$.

**Theorem 3.** *Let* $S$ *be the random variable denoting the number of vertices* $v \in R$ *such that* $\deg_H(v) \geq a$*. Then*

$$\mathrm{E}[S] \geq r \left(1 - e^{-ck + \frac{a-1}{r}} \frac{(ck)^a - 1}{ck - 1}\right)$$

*where the expectation is over the randomness of* $G$ *and* $H$*.*

*Proof.* Let $X_{uv}$ be the indicator variable of the event that the edge $uv$ (note that $u \in L$ and $v \in R$) is in the subgraph that we picked and set $X_v = \sum_{u \in U} X_{uv}$ so that $X_v$ represents the random degree of the vertex $v$ in our subgraph. Because our algorithm uniformly subsamples a uniformly random selection of edges, we can assume that$H$ was generated the same way as $G$ but sampled $c$ instead of $d$ edges for each vertex $u \in L$.

So $X_{uv}$ is a bernoulli random variable. Using the trivial bound $\binom{n}{i} \leq n^i$ on binomial coefficients we get:

$$\begin{aligned}
\Pr[X_v < a] &= \sum_{i=0}^{a-1} \binom{cl}{i} \left(1 - \frac{1}{m}\right)^{cl-i} \left(\frac{1}{r}\right)^i \\
&\leq \sum_{i=0}^{a-1} \left(\frac{cl}{r}\right)^i \left(1 - \frac{1}{r}\right)^{cl-i} \\
&= \left(1 - \frac{1}{r}\right)^{cl-(a-1)} \sum_{i=0}^{a-1} (ck)^i \\
&\leq \left(1 - \frac{1}{r}\right)^{cl-(a-1)} \frac{(ck)^a - 1}{ck - 1} \\
&\leq e^{-ck + \frac{a-1}{r}} \frac{(ck)^a - 1}{ck - 1}
\end{aligned}$$

Letting $Y_v = [X_v \geq a]$, we now see that

$$\mathrm{E}[S] = \mathrm{E}\left[\sum_{v \in R} Y_v\right] \geq r\left(1 - e^{-ck + \frac{a-1}{r}} \frac{(ck)^a - 1}{ck - 1}\right)$$

□

Now, we can perform a similar analysis as in the previous section. In particular, if $ck > a$, then the problem is easy on average though we need $ck$ to get larger than before to be close to optimal. This is in comparison to the trivial estimate of $m$. For a fixed $a$, a random solution gets better as $ck$ increases because the decrease in $e^{-ck}$ more than compensates for the polynomial in $ck$ next to it. However, if $ck < a$, we need to use the trivial estimate of $ckr/a$, and the analysis from the previous section does not extend here.

In both this section and the previous one, $ck$ is the average degree of a vertex $v \in R$ in our chosen subgraph. Basically, what the original analysis said is that if $ck > 1$, then the sampling algorithm will probably cover every vertex in $R$ since the expected degree of each vertex is large. On the other hand if $ck$ is small ($ck < 1$) then the best possible solution is obtained when none of the vertices in $R$ has degree greater than 1. If $ck < 1$, then we do not cover very many vertices in $R$, but we also do not cover many vertices more than once. Since the optimal solution in this case was correspondingly low, our solution was good in the $a = 1$ case. However, when $a < 1$, the fact that our edges are well-dispersed only hurts our solution because we need to concentrate the edges on particular $v \in R$ that will count. The following table shows how large $ck$ needs to be for the solution to be 95% optimal for different values of $a$:

| $a$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $ck$ | 3.00 | 4.74 | 7.05 | 10.01 | 13.48 |

Figure 2: The required $ck$ to obtain 95% optimality

## 4.2 Hierarchical Tree Model

In this section we explore the hierarchical tree model. We will assume that we are given a bipartite graph $G = (L, R, E)$. The vertex sets $L$ and $R$ are the leaf sets of two full binary trees $T_L$ and $T_R$ of depth $D$ where there is a one-to-one correspondence between the subtrees of these two trees. We also assume that each branching in both $T_L$ and $T_R$ splits the nodes evenly into the two subtrees. By abuse of notation, we will use a subtree and its leaf set interchangeably. The trees are fixed in advance, but $G$ is generated probabilistically according to the following procedure. Let $u \in L$ and $T_L^0, \ldots T_L^{D-1}$ be the subtrees it belongs at depths $0, \ldots, D-1$. Also, let $T_R^0, \ldots, T_R^{D-1}$ be the subtrees on the right that correspond to these trees on

the left. We let $u$ make an edge to $d_{D-1}$ of the vertices in $T_R^{D-1}$, $d_{D-2}$ edges to the vertices in $T_R^{D-1} \backslash T_R^{D-2}$ and so on. Each vertex is chosen with uniform probability and we let $d = d_0 + \ldots + d_{D-1}$.

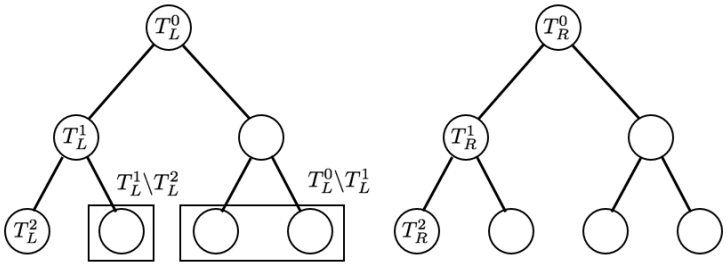


Figure 3: This diagram shows the notation we use for this model and the 1-to-1 correspondance of subtrees

Our goal now is to find a $b-$matching in this graph that is close to optimal in expectation. That is, our degree upper bound on vertices in $L$ is $c$ and the degree lower bound on vertices in $L$ is 1. Let $c = c_0 + \ldots + c_{D-1}$ where the fractions $d_i/c_i$ are all equal to some constant $k$. For simplicity assume that $k$ is an integer. To combine the analysis of the randomness of the algorithm and the randomness of the graph, the algorithm will pick $c_i$ edges uniformly from among the $d_i$ edges going to each level of the subtree. This enables us to think of the subgraph our algorithm outputs as being generated by the graph generation process, but with fewer neighbors selected for each as in the previous sections. With this model and parameters in place, we can prove the following theorem.

**Theorem 4.** *Let $S$ be the subset of edges $v \in R$ such that $\deg_H(v) \geq 1$. Then*

$$E[S] \geq r(1 - \exp(-ck))$$

*where the expectation is taken over $G$ and $H$.*

*Proof.* Let $v \in R$ and let $T_L^{D-1}, T_L^{D-2} \backslash T_L^{D-1}, \ldots, T_L^0 \backslash T_L^1$ be the sets it can take edges from. Since $T_L$ and $T_R$ split perfectly evenly at each node the vertices in these sets will be choosing from $r_{D-1}, r_{D-1}, r_{D-2}, \ldots, r_1$ vertices in $R$ for neighbors respectively, where $r_i$ is the size of subtree of the right tree rooted at depth $i$. Furthermore, each of these sets described above have size $l_{D-1}, l_{D-1}, l_{D-2}, \ldots, l_1$ respectively, where $l_i$ is the size of a subtree of $T_L$ rooted at depth $i$. It follows that the probability that $v$ does not receive any edges at all is at most

$$\begin{aligned}
\Pr[\neg X_v] &= \left(1 - \frac{1}{r_{D-1}}\right)^{c_0 l_{D-1}} \prod_{i=1}^{D-1} \left(1 - \frac{1}{r_i}\right)^{c_{D-i} l_i} \\
&\leq \exp\left(-\frac{l_{D-1}}{r_{D-1}} c_0\right) \prod_{i=1}^{D-1} \exp\left(\frac{l_i}{r_i} c_{D-i}\right) \\
&= \exp\left(-(c_0 + \ldots + c_{D-1})k\right) \\
&= \exp(-ck)
\end{aligned}$$

Since this is an indicator variable, it follows that

$$\mathrm{E}[S] = \mathrm{E}\left[\sum_{v \in R} X_v\right] \geq r\left(1 - \exp(ck)\right)$$

□

Note that this is the same result as we obtained for the fixed degree model in Section 4.1. In fact, the approximation guarantees when $ck \ll 1$ or $ck \gg 1$ hold exactly as before.

The sampling of $H$ can be done algorithmically because we separated out the edge generation process at a given depth from the edge generation process at deeper subtrees. There is no ambiguity as to why an edge is in the underlying graph. That is, if we superimpose $T_L$ and $T_R$, then an edge between $u_l \in L$ and $v_r \in R$ must have come from an edge generated at the lowest common ancestor of $u_l$ and $v_r$. So the algorithm can actually sample intelligently and in the same way that the graph was generated in the first place. Also note that we do not have to assume that the trees $T_L$ and $T_R$ are binary. We only need the trees to be regular and evenly divided at each vertex since the proof only relies on the proportions of the sizes of the subtrees in $T_L$ and $T_R$.

## 4.3 Cartesian Product Model

We can extend the analysis in Section 4.1 in a way orthogonal to the hierarchical tree model as follows. We assume that $L$ has been partitioned into $t$ subsets $L_1, \ldots, L_t$ and that $R$ has been partitioned into $t'$ subsets $R_1, \ldots, R_{t'}$. For convenience, we let $|L_i| = l_i$ and $|R_i| = r_i$. Given this suppose that for each $1 \leq i \leq t$ and each $1 \leq j \leq t'$, $G[L_i, R_j]$ is an instance of the Fixed Degree Model with $d = d_{ij}$. We assume that for all $i$, we have $\sum_{j=1}^{t'} d_{ij} = d$ for some fixed $d$. Also assume that we have fixed in advance $c_{ij}$ for each $1 \leq i \leq t$ and $1 \leq j \leq t'$ that satisfy $\sum_{j=1}^{t'} = c$ for some fixed $c$. To sample $H$ from $G$, we sample $c_{ij}$ neighbors for each $u_i \in L_i$ from $R_i$. Letting $S$ be the set of vertices in $v \in R$ that satisfy $\deg_H(v) \geq 1$, we can show the following:

**Theorem 5.** *With $S$, $G$ and $H$ defined as above, we have*

$$E[S] \geq r - \sum_{j=1}^{t'} r_j \exp\left(-\sum_{i=1}^{t} c_{ij} \frac{l_i}{r_j}\right)$$

*where the expectation is over $G$ and $H$.*

*Proof.* Let $v_j \in R_j$ be an arbitrary vertex and let $X_{v_j}$ be the indicator variable for the event that $\deg_H(v_i) \geq 1$. The probability that none of the neighbors of some $u_i \in R_i$ is $v_j$ is exactly $(1 - \frac{1}{r_j})^{c_{ij}}$. It follows that the probability that the degree of $v_j$ in the subgraph $H[L_i, R_j]$ is 0 is at most $(1 - \frac{1}{r_j})^{c_{ij} l_i}$. Considering this probability over all $R_j$ gives us:

$$\Pr[X_{v_i} = 0] = \prod_{i=1}^{t} \left(1 - \frac{1}{r_j}\right)^{c_{ij} l_i} \leq \exp\left(-\sum_{i=1}^{t} c_{ij} \frac{l_i}{r_j}\right)$$

By linearity of expectation $\mathrm{E}[S] = \sum_{i=1}^{t'} r_i \mathrm{E}[X_{v_i}]$, so it follows that

$$\mathrm{E}[S] \geq \sum_{j=1}^{t'} r_j \left(1 - \exp\left(-\sum_{i=1}^{t} c_{ij} \frac{l_i}{r_j}\right)\right) = r - \sum_{j=1}^{t'} r_j \exp\left(-\sum_{i=1}^{t} c_{ij} \frac{l_i}{r_j}\right)$$

□

This model is interesting because it can capture a broader set of recommendation graphs than the fixed degree model. However, it is difficult to estimate how good a solution will be without knowing the sizes of the sets in the partitions. However, we can note that we obtain the approximation guarantee of $(1-\exp(-ck))$ provided that $l_i/r_j = k$ for all $i$ and $j$ where $k$ is some fixed constant. Another interesting point about this model and the algorithm we described for sampling $H$ is that we are free to set the $c_{ij}$ as we see fit. In particular, $c_{ij}$ can be chosen to maximize the approximation guarantee in expectation we obtained above using gradient descent or some other first order method prior to running the recommendation algorithm to increases the quality of the solution.

### 4.4 Weighted Model

The fixed degree model of Section 4.1 is a simple and convenient model, but the assumption that all recommendations hold the same weight is unrealistic. This motivates fixing the graph to be the complete bipartite graph $K_{l,r}$, and giving the edges i.i.d weights with mean $\mu$. We modify the objective function accordingly, so that we count only the vertices in $R$ which have weight $\geq 1$. If we assume that $ck\mu \geq 1 + \epsilon$ for some $\epsilon > 0$, then naive the solution sampling solution we outlined in Section 4.1 still performs exceptionally well. Letting $S$ be the size of the solution produced by this algorithm we have:

**Theorem 6.** *Let $G = K_{l,r}$ be a bipartite graph where the edges have i.i.d. weights and come from a distribution with mean $\mu$ that is supported on $[0, b]$. If the algorithm from Section 4.1 is used to sample a subgraph $H$ from $G$, then*

$$E[S] = \sum_{v \in R} E[X_v] = r\left(1 - \exp\left(-\frac{2l\epsilon^2}{b^2}\right)\right)$$

*Proof.* For each edge $uv \in G$, let $W_{uv}$ be its random weight, $Y_{uv}$ be the indicator for the event $uv \in H$ and define $X_{uv} = Y_{uv}W_{uv}$. Since weights and edges are sampled by independent processes, we have $\mathrm{E}[X_{uv}] = \mathrm{E}[W_{uv}]\mathrm{E}[Y_{uv}]$ for all edges. Since $c$ edges out of $r$ are picked for each vertex, $\mathrm{E}[Y_{uv}] = \frac{c}{r}$ , so $\mathrm{E}[X_{uv}] = \frac{c}{r}\mu$. Therefore, the expected weight coming into a vertex $v \in R$ would be

$$\mathrm{E}[X_v] = \sum_{u \in L} \mathrm{E}[X_{uv}] = \frac{cl\mu}{r} = ck\mu$$

However, $X_{uv}$ for each $u$ are i.i.d random variables. Since by assumption $ck\mu = 1 + \epsilon$, by a Hoeffding bound we can obtain:

$$\Pr[X_v \leq 1] = \Pr[X_v - \mathrm{E}[X_v] \geq \epsilon] \leq \exp\left(-\frac{2l\epsilon^2}{b^2}\right)$$

By linearity of expectation we can now get the result in the theorem

$$\mathrm{E}[S] = \sum_{v \in R} \mathrm{E}[X_v] = r\left(1 - \exp\left(-\frac{2l\epsilon^2}{b^2}\right)\right)$$

□

There are two things to note about this variant. The first is that since the variables $X_v$ are negatively correlated, our results in 5 can be readily extended to the results of this section. The second is that the condition that $W_{uv}$ are i.i.d is not necessary to obtain the full effect of the analysis. Indeed, the only place in the proof where the fact that $W_{uv}$ are i.i.d is when we argued that $X_{uv}$ is large with high probability by a Hoeffding bound. For the bound to apply, it's sufficient to assume that $W_{uv}$ for all $v$ are independent. In particular, it's possible that $W_{uv}$ for all $u$ are inter-dependent. This allows us to assume an weight distribution that depends on the strength of the recommender and the relevance of the recommendation separately.

## 5 Worst-Case Approximation

The results in the previous section concentrated on producing nearly optimal solutions in expectation. In this section, we will show that it is possible to obtain good solutions regardless of the model that generated the recommendation graph. As usual we let $G = (L, R, E)$ be a bipartite graph on which we would like to solve the (c,a)-graph recommendation problem and consider the following greedy algorithm. Consider each vertex in $R$ in some arbitrary order and if there is some $v \in R$ that has $a$ neighbors in $L$ all of which have degree $< c$, add the edges to these neighbors to $H$. If there are any ties about the nodes to be picked either in the selection of $v$ or its neighbors, we can break ties arbitrarily. This algorithm has the following approximation guarantee

**Theorem 7.** *The greedy algorithm described above achieves a* $1/(a+1)$*-approximation ratio.*

*Proof.* Let $R_{GREEDY}, R_{OPT} \subseteq R$ be the set of vertices that have degree $\geq a$ in the greedy and optimal solutions respectively. Note that any $v \in R_{OPT}$ along with neighbors $\{u_1, \ldots u_a\}$ forms a set of candidate edges that can be taken by the greedy algorithm. So we can consider $R_{OPT}$ as a candidate pool for $R_{GREEDY}$. Each move that the greedy algorithm makes might make some of the candidates infeasible, but as long as the candidate pool is not depleted, the greedy algorithm can continue adding vertices to its solution. Each time the greedy algorithm claims some vertex $v \in R$ with edges to $\{u_1, \ldots, u_a\}$, we have obviously have to remove $v$ from the candidate pool. If any $u_i$ was saturated (i.e. had degree $c$) in the optimal solution, we would also need to remove an arbitrary vertex $v_i \in R$ adjacent to $u_i$ in the optimal solution. In other words, by using an edge of $u_i$, we force it to not use an edge it used to some other $v_i$, which might cause the degree of $v_i$ to go below $a$. (Note that the greedy algorithm does not actually have to be aware of the structure of optimal solution for this type of bookkeeping to go through.) Therefore, at each step of the greedy algorithm, we have to remove at most $a+1$ vertices from the candidate pool. Since our candidate pool has size $OPT$, the greedy algorithm cannot stop before it has added $|OPT|/(a+1)$ vertices to the solution. □

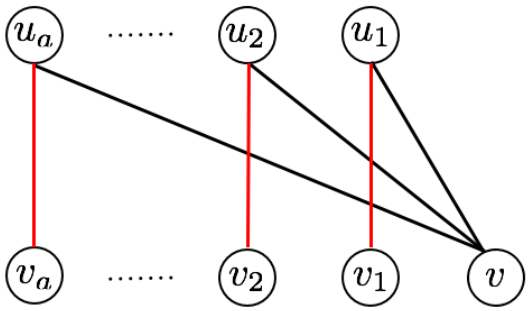


Figure 4: This diagram shows one step of the greedy algorithm. When $v$ claims edges to $u_1, \ldots, u_a$, it potentially removes $v_1, \ldots, v_a$ from the pool of candidates that are avaiable. The potentially invalidated edges are shown in red.

There are several things to note about this algorithm. The first is that the solution the greedy algorithm outputs is dependent on the order the vertices in $R$ are processes. It may be possible to improve the expected approximation ratio by permuting these vertices randomly before the algorithm runs. The second is that this approximation guarantee is as good as we can expect. In particular, if we set $a = 1$ then we obtain the familiar 1/2-approximation for matchings. It might be possible that some or even all the cases of the graph recommendation problem admit $(1+\epsilon)$ approximations in $O(n \cdot \text{polylog}(n))$ time, but this greedy algorithm likely represents the best we can do with minimal effort. Finally, we can use this worst case approximation algorithm to benchmark the random sampling algorithms we described in the previous section.

# 6 Comparing the Worst-Case and Average-Case Approximations

In this section, we show that the the results of Section 4.1 that hold in expectation also hold in the worst case with high probability using concentration bounds.
We can convert the expectation results to a lower bound that holds with high probability in two ways. First, recall that Markov's inequality states that for any non-negative random variable $X$ and any $a \geq 1$, we have $\Pr[X \geq a\mathrm{E}[X]] \leq \frac{1}{a}$. We can apply this to the random variable $r - S$ with $a = \frac{1}{2}$ to obtain:

**Theorem 8.** *The random sampling algorithm produces a solution of size at least* $r(1 - 2\exp(-ck))$ *with probability at least* $1/2$*,*

While this gives us a lower bound, we can do better using Chernoff bounds. While Chernoff bounds are usually stated for independent variables, the bound bound below holds for non-positively correlated variables.

**Theorem 9.** *Let $X_1, \ldots, X_n$ be non-positively correlated variables. If $X = \sum_{i=1}^{n} X_i$, then for any $\delta \geq 0$*

$$Pr[X \geq (1+\delta)E[X]] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{E[X]}$$

Now note that the variables $1 - X_v$ for each $v \in R$ are non-positively correlated. In particular, if $N(v)$ and $N(v')$ are disjoint, then $1 - X_v$ and $1 - X_{v'}$ are independent. Otherwise, $v$ not claiming any edges can only increase the probability that $v'$ gets an edge from any vertex $u \in N(v) \cap N(v')$, so the variables would be negatively correlated. Setting $\delta \approx 1.39$ in the theorem above now proves the following:

**Theorem 10.** *The random sampling algorithm produces a solution of size at least $r(1 - 2\exp(-ck))$ with probability at least $(e/4)^{r\exp(-ck)}$.*

That is, the probability of producing a constant factor approximation decreases exponentially with $r$.
With these high probability bounds in place, it is instructive to compare the performance of the random sampling algorithm with that of the $c = 1$. With high probability, the approximation ratio of the randomized algorithm is $(1 - \exp(-ck))/(1 - \exp(-dk))$ while the approximation ratio of the greedy algorithm is always at least $1/2$. In the following graphs $f \geq 1$ represents a value such that $d = fc$ and we use the values $k = 0.1$ and $k = 0.2$ respectively since the case that is practical for our purposes is when $l \ll r$. In the graphs below, the red curve shows the approximation ratio for the model used in Section 4.1 and the blue curve shows the worst case approximation ratio described in section 5. As we would expect, the sampling algorithm performs outperforms the greedy algorithm significantly when $k$ and more importantly $c$ gets larger.

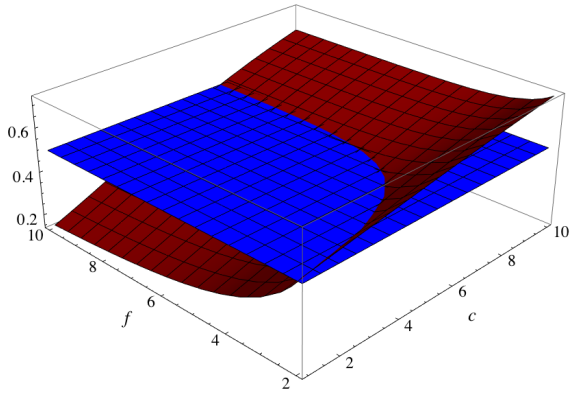


Figure 5: Approximation ratios when $k$=0.1

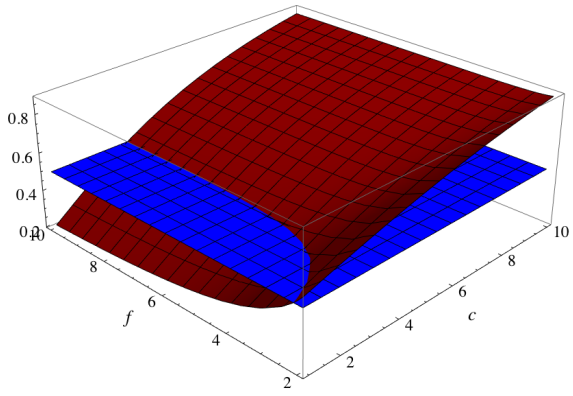


Figure 6: Approximation ratios when $k$=0.2

# 7 Future Directions

In this paper we proposed several different models for explaining how recommendation graphs arise probabilistically and how optimization problems on these graphs can be solved, but there is much more that can be done.

1. We only investigated the maximum cardinality version of our problem. However, a recommendation on page that generates 1 million views is much more valuable than a recommendation on a page that generates 10 thousand views. Our problem can be quite easily generalized to involve weights on edges and it's quite valuable to explore this generalization.

2. In practice, there might exist several different recommendation graphs based on different features. For example, two people might be related because they went to the same school, or because they live in the same city, or because they would complete a large number of triangles, etc. It's worthwhile to investigate how such graphs can be combined into one, or how an optimization problem can be solved using all such recommendation graphs simultaneously.

3. We should devise metrics that can evaluate how well a recommendation graph fits a given model and conduct some parameter fitting experiments to see how well actual recommendation graphs which arise in practice fit our models.

4. We should implement the sampling and the greedy algorithms given in the paper to see if they can solve to near optimality the graph recommendation problems that arise in practice.