The Lazy Engineer and Random Recommendation Subgraphs

Arda Antikacioglu, R. Ravi, Srinath Sridhar

June 30, 2013

Abstract

The utility of many of the popular sources of information in the web such as YouTube, FaceBook, and Quora, as well as many store catalog sites, arises from the recommendations they supply to enable a surfer to continue to discover new and useful information or products. Such sites usually have a large list of candidate pages that serve as good recommendations from the current page, based on topic analysis and relevance. If every page is a node, these candidate recommendations can be denoted as directed arcs in a large directed graph. The recommendation subgraph problem is loosely to choose a subgraph of bounded outdegree (denoting the limited page space to display the recommendations) that allows "efficient" navigation using the subgraph links over the whole repository.

In a simplification of the problem, the set of pages is divided into the few popular pages into which surfers arrive at the site, and the remaining large number of unknown pages that can be potentially recommended from the popular pages. The resulting natural bipartite graph of recommendations from the popular to the undiscovered pages is the candidate supergraph, from which a subgraph that has bounded outdegree on the popular pages must be chosen. The objective of reaching as many undiscovered pages as possible from any of the popular pages in the subgraph leads to a variant of bipartite matching. Maximizing the number of pages that are recommended by at least two or more popular pages leads to a much harder problem.

We describe our work in solving such recommendation subgraph problems in practice at web scale. We observe the startling effectiveness of a natural greedy method that would be proposed by a lazy engineer in place of bipartite matching for this problem. To explain this, we investigate the cases when the candidate supergraph is a random subgraph under two variations: the "d-out" or fixed-degree version where every node on the left has exactly d random neighbors on the right, as well as the standard Erdös-Renyi model with expected degree d on the left hand side. We show that for most reasonable cases of the input, the lazy method gives solutions with value very close to optimal, and hence delineate the parameter ranges when it pays for the engineer not to be lazy.

We then extend the fixed-degree random graph model in many directions: one where the pages are situated in a natural hierarchical taxonomy (such as a topic ontology in Quora, or a product hierarchy in a store catalog), another where varying densities are allowed between different subgraphs, and a third where the recommendation edges are weighted (typically according to the traffic that the particular link is likely to generate) and the degree constraints are on the weighted degree. In all these cases, we identify conditions under which the lazy engineer is very effective. We conclude with web-scale experimental results that motivate and validate our proposed models, as well as compare the effectiveness of various heuristic methods that the lazy engineer would have then thought up in his freed up time.

1 Introduction

One of the great benefits of the web as a useful source of hyperlinked information comes from the careful choices made in crafting the recommendations that link a given page to closely related pages.

Even though this advantage was identified well before the www was in place by Bush [**?**], it continues to persist even in Web 2.0 systems that generate web pages of combined information, where smooth navigation is enabled by carefully chosen recommendations to closely related pages. Thus, the presence of recommendations is an integral feature of several popular websites that are known to have high user engagement as reflected in long user sessions. Examples range from entertainment sites like YouTube that recommends a set of videos on the right for every video watched by a user, information sites like Quora that recommends a set of related questions on every question page, to retail sites like Amazon that recommends similar products on every product page.

While recommendations are important, they are implemented typically by finding related items to display at runtime. That is, when a user lands on an item the recommendation systems decide online the set of other items to display. Such systems have the problem that a global analysis of the performance of the recommendations are hard. E.g., it is almost impossible to know if there is an item that was not recommended by even one other item. As the size of the repository grows, doing such a real-time computation becomes less feasible and might lead to deterioration in the quality of the recommended pages. This motivates the strategy of solving the recommendation problem off-line, where we pre-compute the set of recommendations for every item. Such a system can be built in two stage: the first stage decides on recommendations for each item purely based on relevance and retrieves a large candidate set of d items to show. The second stages prunes it to $c < d$ items such that globally we ensure that the resulting recommendation graph is 'good'. For instance, we might want to ensure that the resulting graph minimizes the number of items that was not recommended by a single other item.

We can represent this notion of recommendations by using a directed graph. A vertex is simply an item and a directed edge (u, v) is a recommendation from u to v. Under this graph model for the above problem, if $c = 1$ and we require the chosen recommendation subgraph be strongly connected then our problem reduces to Hamilton cycle, an NP-complete problem.

1.1 Bipartite Recommendation Subgraphs

We can generalize the above graph recommendation problem by using a directed bipartite graph. Website owners might not be interested on all the items on their site. We can use one partition L (say, the one in the left) to represent the set of items for which we are required to suggest recommendations. We can use the other partition R (on the right) to represent the set of items that are potentially recommended. Note that a single item can be represented in both L and R if needed. We will use this representation to formulate all our results. Now the input to the problem is this directed graph G where each vertex has d recommendations (thanks to the first stage of the two-stage recommendation back-end system described above). The output required is a subgraph G' where each vertex in L has $c < d$ recommendations. The simplest goal is to minimize the number of vertices that have in-degree less than an integer quantity a. We call this the (c, a)-graph recommendation problem. Note that if $a = c = 1$ this is simply the problem of maximum bipartite matching [**?**]. If $a = 1$ and $c > 1$, this is still a degree-bounded subgraph problem that can be converted to a polynomial-time solvable matching problem.

Over the past few years, a subset of us have implemented such recommendation subgraph algorithms in cutting-edge web-technology companies including Google, Facebook and BloomReach. There are two key hurdles in making these recommendation subgraph back-end systems practical. The first is that the method used must be very simple to implement, debug and deploy. The second is that the method must scale gracefully. Matching algorithms require linear memory and super-linear run-

time neither of which scale well. Note that the Facebook graph has over a billion vertices [] and hundreds of billions of edges [], YouTube has XXX videos and YYY recommendations [] etc. Even an ecommerce website with 100M product pages and 10 recommendations per product would require over 100GB in main memory to run these algorithms. Since using clusters of single machines with such high memory is prohibitively expensive, MapReduce [] jobs are needed to solve these problems. However, such Map-reduce for graph problems are notoriously hard and we are back to building a complicated system.

Our work with real-life web systems suggest that typical values for c range from 5 to 20, while the requirement a is typically one, and sometimes slightly larger in the range 2 to 5. The ratio of the size of undiscovered pages on the right to the popular pages on the left is of the order of ten to hundred. it is instructive to check the performance of the analyzed algorithms for these typical values of these parameters.

1.2 Algorithms and Analyses

The Lazy Engineer These reasons motivated the authors to investigate the "lazy" approach of choosing a very simple (any) set of recommendation to see if they would produce near optimal solutions at least under realistic scenarios in practice. Our initial goal was to find simple relationships between the parameters of the problem and compute thresholds that dictate the need for different, more sophisticated algorithms. In practice, this would immediately imply that a very simple back-end system can be built in many cases without the need for a complicated algorithm. If the thresholds provided by our analysis are insufficient in quality, then the designer can consider implementing the classical algorithms.

Recommendation Graph Models The setting for investigating the effectiveness of the lazy engineer is provided by using a random graph model for the recommendation supergraph. Since the lazy algorithm involves choosing *any* set of c recommendations, the natural random graph model that permits analysis of this method is the *fixed-degree* model where every node on the left has recommendations to a random subset of size exactly d nodes in the right.

We study this model in Section 2.1. Our main result identifies the range of parameters involving $c, a, l = |L|$ and $r = |R|$ where the lazy algorithm is very effective. In addition to showing that it is a $(1 - \frac{1}{e})$-approximation algorithm in expectation, we also get much better bounds for the expected performance for a wide range of realistic parameters and also prove high probability bounds on its performance.

The Greedy Algorithm While the lazy algorithm chooses any set of c recommendations, the natural greedy algorithm will need some work: scanning the nodes on the right that must be discovered, we look to see if there are a neighbors from the left that have not exhausted their bound of c edges in the subgraph, and if so, use them to add this node to the discovered set. We study this algorithm in Section 2.2. We show the easy result that Greedy gives a $\frac{1}{a+1}$-approximation, and also do an average case analysis. Hwever, we use the usual Erdös-Renyi model rather than the fixed-degree model for this analysis.

In the subsequent Section 2.3, we compare and plot the worst-case performance of Greedy against the average case expected performance of the random or lazy algorithm, as a basis for our later computational comparisons in a similar vein.

Optimal Recommendation Subgraphs Finally we turn to the question of whether all nodes on the right can be covered (at least a times) by a recommendation subgraph: we say that there is a perfect (c, a)-recommendation subgraph in this case. Under the usual Erdös-Renyi model, we use existing results on the existence of a perfect matching to characterize the edge probability

over which there exist such perfect recommendation subgraphs, first for the case when $a = 1$, and then for the case of more general $a > 1$, by using a subset partitioning method for the analysis. By relaxing the condition of optimality of matching (i.e. perfect matchings) to disallowing short augmenting paths in these methods, we also turn the above results into $(1 - \epsilon)$-approximation algorithms trading off the running time, for appropriately dense random graphs as before.

1.3 New Recommendation Graph Models

The analysis of the various algorithms suggests that there may be more general settings where the lazy and greedy algorithms perform well. We use our experience in working with practical web systems to propose three such models that generalize the fixed-degree model by taking into account (i) a natural hierarchy in the organization of the pages, (ii) a partition of pages into disjoint categories and varying density of connections between different categories, and (iii) a weighting of the edges of the recommendation supergraph based on the traffic generated along each of these links.

A Hierarchical Model. Most information sources organize their pages according to a natural taxonomy such as an ontology of topics, or a classification of retail products or information records according to their types. In this generalization, we assume that there is such a natural hierarchy classifying the nodes in both sides of the bipartite recommendation supergraph.

For example, a page about random graph models in Quora might be under a cluster of pages about graphs in general, which in turn might come under another larger superset of pages about general discrete models, giving a two level hierarchy. A recommendation from a popular page on, say the fixed-degree model, might go to an obscure page on some other graph model, or to an obscure page on discrete structures that is not a graph model. Typically, the fraction of recommendations going to topics not in one's own cluster will decrease exponentially as we move higher up in the taxonomy. In Section 3.1, we show how the analysis of our main result for the fixed-degree model extends to this case as well.

A Cartesian Product Model. Our second generalization is motivated by the case when the pages are partitioned into disjoint categories in both sides of the bipartite graph. The density of random recommendations in the supergraph may vary across different pairs of categories. To smoothly generalize the results from before, we assume that despite the variation, every node in the left has the same final fixed degree while this bound can be distributed differently between the various categories on the right hand side depending upon which category on the left side the node belongs to. Our analysis in Section 3.2 again extends our basic result about the effectiveness of the lazy algorithm to this model.

Weighted Links. In the last generalization, we assume that every link in the candidate supergraph of recommendations is weighted by a fraction that represents the normalized traffic that will be generated by including this link. Assuming that we can still direct only a total of c units of normalized traffic from the left hand side node, we can still investigate the question of maximizing the number of right side nodes that have normalized recommending traffic summing to at least one. In Section 3.3, we show the parameters under which the lazy algorithm is effective for this model.

1.4 Computational Results

We performed extensive computational testing of the two algorithms we analyze: the lazy random choice algorithm, and the greedy, as well as some variants where the greedy algorithm truncates its search for matching nodes from the left after examining a certain small number of neighbors from the left. In all our testing that were conducted up to web-scale graph sizes, the greedy algorithm

almost always outperforms all the other methods, while the lazy method, as predicted by our analysis above, are very effective for a large swath of parameter ranges.

2 Algorithms for Recommendation Subgraphs

In this section, we analyze the lazy algorithm of choosing any set of c recommendations, and the slightly more interesting greedy algorithm for finding a (c, a)-recommendation subgraph. To do this, we first introduce the fixed-degree random graph model for the candidate supergraph of recommendations.

2.1 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 $H \subseteq G$ 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.

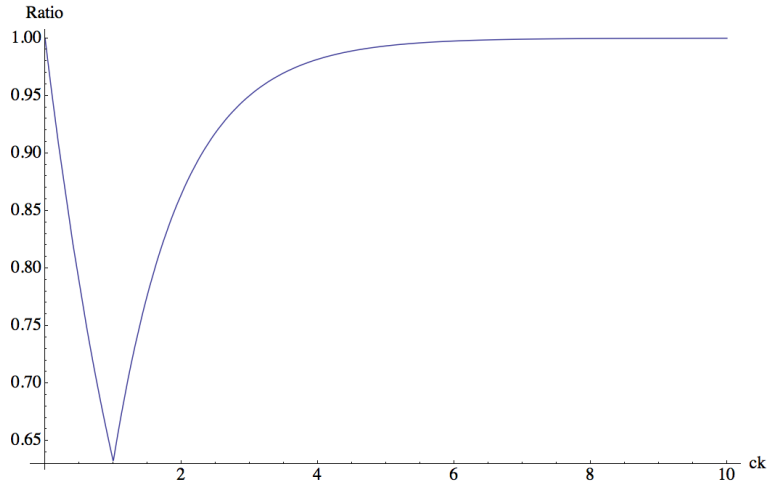

Figure 1: Approximation Ratio as a function of ck

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:

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

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 before. 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 cl. 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, in the more realistic case $ck < a$, we need to use the trivial estimate of ckr/a, and the analysis from the previous section does not extend here.

In both this analysis and the previous one, ck is the average degree of a vertex $v \in R$ in our chosen subgraph. The original analysis showed 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 nodes in R that will eventually count in the objective. 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

We close out this section by showing that the main result that holds in expectation also hold with high probability using concentration bounds. While Chernoff bounds are usually stated for

independent variables, the variant below holds for any number of pairwise non-positively correlated variables.

Theorem 4. *[] 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 $\delta = 1$ in the theorem above now proves the following:

Theorem 5. *The random sampling algorithm produces a solution of size at least $r(1-2\exp(-ck))$ with probability at least $1 - (e/4)^{r\exp(-ck)}$.*

That is, the probability of producing a constant factor approximation decreases exponentially with r.

2.2 The Greedy Algorithm

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 subgraph. 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.
We analyze the following natural 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.

Theorem 6. *The greedy algorithm achieves a $1/(a+1)$-approximation ratio for the (c, a)-graph recommendation problem.*

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. □

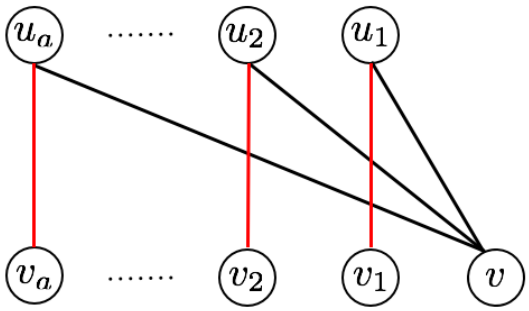

Figure 3: This diagram shows one step of the greedy algorithm. When v claims edges to $u_1, \ldots, u_a$, it potentially removes $v_1, \ldots, v_a$ from the pool of candidates that are avaiable. The potentially invalidated edges are shown in red.

There are two 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 processed. 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 of the greedy algorithm for matchings.

Theorem 7. *Greedy gives sublinear error for $G_{n,n,p}$ if pn is an arbitrarily slow growing function.*

Proof. Note that if edges are generated uniformly, then we can consider the graph as being revealed to us one vertex at a time as the greedy algorithm runs. In particular, consider the event X_i that the greedy algorithm matches the $(i+1)^{th}$ vertex it inspects. While, X_{i+1} is dependent on $X_1, \ldots, X_i$, the worst condition for X_{i+1} is when all the previous i vertices were by the same vertices in L, which are now not available for matching the $(i+1)^{th}$ vertex. The maximum number of such invalidated vertices is at most $\lceil i/c \rceil$. Therefore, the probability that none of the $n - \lceil i/c \rceil$ vertices have an edge to this vertex is at most $(1-p)^{n-i/c}$. Summing over all the X_i and using the linearity of expectation, we obtain the following inequality:

$$\sum_{i=0}^{n-1} \mathrm{E}[\neg X_i] \leq \sum_{i=0}^{n-1} (1-p)^{n-i-1} = \sum_{j=0}^{n-1} (1-p)^j \leq \sum_{j=0}^{\infty} (1-p)^j = \frac{1}{p}$$

That is, the expected size of the solution found by the greedy algorithm is $n - 1/p$. □

2.3 Comparing the Worst-Case and Average-Case Approximations

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 2.1 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.

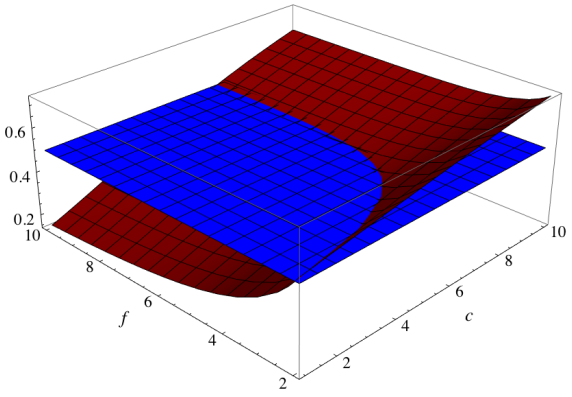

Figure 4: Approximation ratios when k=0.1

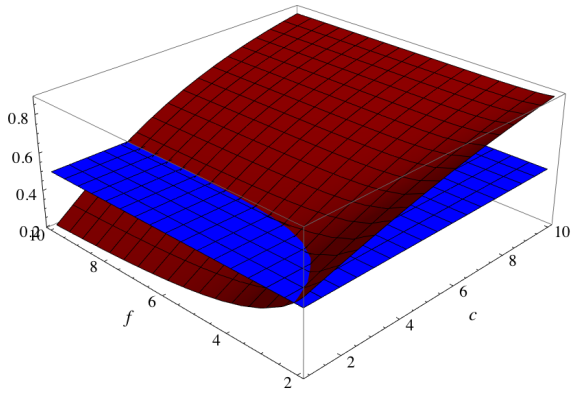

Figure 5: Approximation ratios when k=0.2

2.4 Existance of Optimal Recommendation Subgraphs

Let $G = (L, R, E)$ be a bipartite graph. We define a *perfect* (c, a)-recommendation on G to be a subgraph H such that $deg_H(u) \leq c$ for all $u \in L$ and $deg_H(v) = a$ for $\min(|R|, |L|c/a)$ of the $v \in R$. In this section we will prove a sufficient condition for perfect (a, a)-recommendation subgraphs to exist in a bipartite graph where edges are sampled uniformly and independently with probability p. Our result relies on a well-known characterization of perfect matchings [**?**]:

Theorem 8. *Let $G = (L, R, E)$ be a bipartite graph with $|L| = |R| = n$ and suppose that G is drawn from $G_{n,n,p}$. If $p \geq (\log(n) - \log\log(n))/n$, then $\lim_{n\to\infty} \Pr[G \text{ has a perfect matching}] \to 1$.*

We will prove that a perfect (c, a)-recommendation subgraph exists in random graphs with high probability by building it up from a matchings which must exists with high probability if p is sufficiently high. In the next theorem, we show that p only needs to be taken $\Theta(\log(n))$ for this plan of attack to succeed. Furthermore, this decomposition into matchings motivates a an algorithm that can return a solution that comes within $(1-\epsilon)$ of the optimal solution and can be run in almost linear time. While the theorem is stated for the case when $c < a$, but it applies equally well to the $a < c$ case by swapping the roles of L and R in the proof.

Theorem 9. *Let $G = (L, R, E)$ be a random graph and let $|L| = l$, $|R| = r$ and $k = l/r$. If each edge of G is sampled with probability $p \geq c(\log(l) - \log\log(l))/l$ and $kc \geq a$, then*

$$\Pr[G \text{ has a perfect } (c, a)\text{-recommendation subgraph}] \to 1$$

as $l, r \to \infty$ together.

Proof. Note that given the size and the degree constraints of L, at most lc/a vertices in R can obtain degree a in a (c, a)-recommendation subgraph. We can therefore restrict R to an arbitrary subset R' of itself of size lc/a. Next, we pick an enumeration of the vertices in $L = \{u_0, \ldots, u_{l-1}\}$ put these vertices into a subsets by defining $L_i = \{u_{(i-1)l/a}, \ldots, u_{(i+c)l/a-1}\}$ for each $1 \leq i \leq a$. Note both R' and all of the L_i have size lc/a.

Using these new sets we defined, we define the graphs G_i on the bipartitions given by (L_i, R'). Since the sets L_i are intersecting, we cannot define the graphs G_i to be induced subgraphs. However, note that each vertex $u \in L$ falls into exactly c of these subsets. Therefore, we can uniformly assign each edge in G to one of c graphs among $\{G_1, \ldots, G_a\}$ it can fall into, and make each of those graphs a

random graph. In fact, while the different G_is are coupled, taken in isolation we can consider any single G_i to be drawn from the distribution $G_{lc/a,lc/a,p/c}$ since G itself was drawn from $G_{l,r,p}$. Since $c/a < 1$ and $p/c \geq (\log(l) - \log\log(l))/l$ by assumption, we can now say that

$$\Pr[G_i \text{ has no perfect matching}] = o(1)$$

Considering a to be fixed, by a union bound we can now conclude that except for a $o(1)$ probability, each one of the G_is has a perfect matching. By superimposing all of these perfect matchings, we can see that every vertex in R' gets degree a. Since each vertex in L is in exactly c of the L_is, each vertex in L gets a degree a. It follows that except for a $o(1)$ probability, there exists an (c, a)-recommendation subgraph in G. □

Using this result, we can come up with an approximation algorithm that can deliver an $(1 - \epsilon)$ approximation to the (c, a)-recommendation problem almost linear time in randomly sampled dense graphs.

Theorem 10. *Let $G = (L, R, E)$ be a graph where the edges are independently and uniformly with probability at least $p \geq (\log(|L|) - \log\log(|L|))/|L|$. There exists an algorithm that can with high probability output a $(1 - \epsilon)$-approximation in $O(|E|a^2/\epsilon)$ time.*

Proof. As in the proof of the previous theorem, we can arbitrarily restrict R to a subset of size $|L|c/a$ and divide L into sets $L_1, \ldots, L_a$ such that each vertex in L is contained in exactly c of the sets. Using the previous theorem, we know that each of the graphs G_i has a perfect matching with high probability. These perfect matchings can be approximated to a $1 - \epsilon/a$ factor by finding matchings that don't have augmenting paths of length $\geq 2a/\epsilon$. This can be done for each G_i in $O(|E|a/\epsilon)$ time. Furthermore, the union of unmatched vertices makes up an at most $a \cdot (\epsilon/a)$ fraction of R'. This proves the result. □

While this result is a heavier weight solution, rephrasing the problem in terms of matchings enables us to take advantage of the well-understood combinatorial properties of matchings.

3 More Expressive Models of Recommmendation Graphs

Having described the fixed-degree model in detail above, we can now use it as a template for the analysis of three more richer and more expressive models. In Section 3.3, we generalize the fixed degree model to a model where the graph is fixed and the edges can weights. In Section 3.1, we analyze the *hierarchical tree model* which aims to model recommendation trees that arise from relationships in a product hierarchy. Finally, in Section 3.2 we study the *cartesian product model* which aims to model recommendation graphs which arise from combining the relevancy advice of several different algorithms.

3.1 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. As in the previous sections, we set $|L|/|R| = k$, but we also note that this ratio still holds if we take any subtree on the left and its corresponding subtree on the right. 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 $d = d_0 + \ldots + d_{D-1}$.

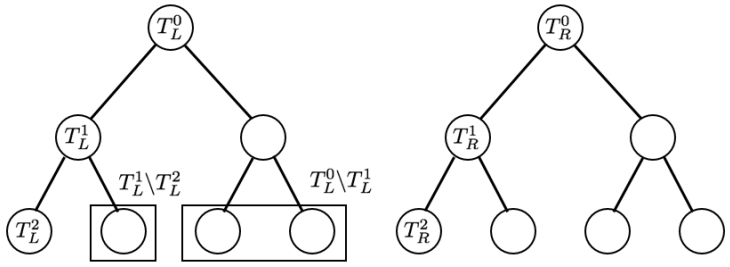

Figure 6: This diagram shows the notation we use for this model and the 1-to-1 correspondance of subtrees

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 $c = c_0 + \ldots + c_{D-1}$ similarly to how we defined d. 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 11. *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 2.1. 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.

3.2 Cartesian Product Model

We can extend the analysis in Section 2.1 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 $\sum_{j=1}^{t'} c_{ij} = c$ for all i 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 12. *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 subgraphs 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.

3.3 Weighted Model

The fixed degree model of Section 2.1 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 μ. We modify the objective function accordingly, so that we count only the vertices in R which have weight ≥ 1. If we assume that $ck\mu \geq 1 + \epsilon$ for some $\epsilon > 0$, then naive the solution sampling solution we outlined in Section 2.1 still performs exceptionally well. Letting S be the size of the solution produced by this algorithm we have:

Theorem 13. *Let $G = K_{l,r}$ be a bipartite graph where the edges have i.i.d. weights and come from a distribution with mean μ that is supported on $[0, b]$. If the algorithm from Section 2.1 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 2.3 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.

4 Future Directions

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

1. In practice, there might exist several different recommendation subgraphs 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 subgraphs simultaneously.

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

3. 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.