The Lazy Engineer and Recommendation Subgraphs*

Arda Antikacioglu[†], R. Ravi[‡] and Srinath Sridhar[§]

Abstract

Recommendations are central to the utility of many of the popular websites such as YouTube, Facebook and Quora, as well as many e-commerce store websites. Such sites usually have a large list of candidate pages that could serve as recommendations from the current page based on relevance. To inter-connect the website for efficient traversal it is critical to choose a few recommendations on each page from the list of all candidates.

In our formulation of the problem, the set of pages on the site is divided into a few popular pages where surfers arrive and the remaining large number of pages that should be recommended from the popular pages. The natural bipartite graph of potential recommendations is a candidate supergraph. Given the limited space to display recommendations a subgraph that has bounded outdegree c on the popular pages must be chosen. Our goal is to maximize the number of undiscovered pages that have indegree at least some target a. We introduce and study this (c, a)-recommendation subgraph problem.

Solving such problems optimally at web-scale involves writing distributed matching algorithms. Instead, in this work, we study the effectiveness of a lazy engineer solving the recommendation subgraph problem. We investigate the cases when the candidate supergraph is a random graph under two models: the fixed-degree model 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 side. We show that for most reasonable parameters of the models, the lazy engineer would have found solutions very close to optimal. We further show the conditions under which a perfect recommendation subgraph, a generalization of perfect matchings, exists. Lastly, to more realistically model web graphs, we propose generalizations of the random graph models using topic taxonomies, varying subgraph edge densities and edge weights that capture the strength of recommendations. Surprisingly, the lazy engineer would still have found near optimal solution under different realistic parameters, which we validate with some computational testing on simulated models.

*This work is supported in part by an internship at BloomReach Inc.

[†]Department of Mathematics, Carnegie Mellon University; aantikac@andrew.cmu.edu

[‡]Tepper School of Business, Carnegie Mellon University; ravi@cmu.edu

[§]BloomReach Inc; srinath@bloomreach.com

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 page to closely related pages. Though this advantage was identified well before the www was in place by Bush [**?**], it continues to persist even today. The presence of recommendations is an integral feature of several popular websites that are known to have high user engagement and long sessions. Examples range from entertainment sites like YouTube that recommend a set of videos on the right for every video watched by a user, information sites like Quora that recommend a set of related questions on every question page, to retail sites like Amazon that recommend similar products on every product page. Recent estimates [**?**] attribute up to a third of the sales in Amazon and three-quarters of new orders in Netflix are due to precisely choosing recommendations.

1.1 Recommendation Subgraphs. Recommendation systems [**?**, **?**, **?**] start by finding a large set of related candidate items for each item (or page) using relevance. In this work, we assume d such related candidates are available per page and our goal is to analyze how to prune the set to $c < d$ recommendations such that globally we ensure that the resulting recommendation subgraph can be traversed 'efficiently' by the user. We represent recommendations by using a digraph where the vertices are items and a directed edge (u, v) is a recommendation from u to v. Note that, under this model, if $c = 1$ and we require the chosen recommendation subgraph to be (strongly) connected, a feasible solution is a Hamilton cycle which is NP-complete to find [**?**].

We generalize the graph recommendation model by using a bipartite digraph, where one partition L represents the set of items for which we are required to suggest recommendations and the other partition R represents the set of items that can be potentially recommended. If needed, the same item can be represented in both L and R. As before, the input to the problem is the graph where each L-vertex has d recommendations. Given the space restrictions to display recommendations, the output is a subgraph where each vertex in L has $c < d$ recommendations. The goal is to maximize the number of vertices that have in-degree at least a target integer a. We introduce and study this (c, a)-recommendation subgraph problem in this work. Note that if $a = c = 1$ this is simply the maximum bipartite matching problem [**?**]. If $a = 1$ and $c > 1$, we obtain a b-matching problem, that can be converted to a bipartite matching problem [**?**].

Motivation from Practice. Over the past few years, the first and third authors have implemented complex graph algorithms in cutting-edge web-technology companies including Google, Facebook and BloomReach. There are two key requirements in making such graph algorithms 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 with larger sizes.

Matching algorithms require linear memory and super-linear run-time which does not scale well. For example, the Facebook graph has over a billion vertices [**?**] and hundreds of billions of edges. A typical e-commerce website with 100M product pages and 100 recommendation candidates per product would require easily over 160GB in main memory to run matching algorithms; this can be reduced by using graph compression techniques but that adds more technical difficulties in development and maintenance. In practice offline problem instances are solved by using distributed computing such as map-reduce [**?**]. However, efficient map-reduce algorithms for graph problems are notoriously difficult and complicated.

We now describe typical web graph characteristics by discussing the sizes of L, R, c and a in practice. It is well known that, in most websites, a small number of 'head' pages contribute to a significant amount of the traffic while a long tail of the remaining pages contribute to the rest [**?**, **?**]. As demonstrated by a prior measurement [**?**] it is not unreasonable to expect 50% of site traffic to be contributed by less than 1% (a few thousands) of the web pages while a large number of tail pages (a few hundred thousand) contribute the other half. This implies that in practice L can be up to two orders of magnitude smaller than R. By observing recommendations of Quora, Amazon, YouTube and our own work, typical values for c range from 3 to 20 recommendations per page. Values of a are harder to nail down but it typically ranges from 1 to 10.

1.2 Main Results. We initiate the study of structural engineering of effective recommendation systems by formulating and solving them as subgraph problems. We offer very good bounds on the performance of algorithms that a lazy engineer would prefer (random choice, greedy) under a natural new fixed-degree model; we extend the analysis of perfect matchings in random graphs to show the range of values of the graph density for which perfect recommendation subgraphs exist; motivated by empirical recommendation supergraphs that we analyze, we propose three new random graph models (hierarchical, cartesian product and weighted) extending our fixed-degree model to capture these real-life scenarios, and extend our analysis of the random algorithm to all these models. We now provide a short overview of these results.

The Lazy Engineer. The implementation challenges mentioned above motivated us to investigate simple algorithms that are practical and can be implemented at very large scale. First, we simply analyze choosing a random c out of d edges and investigating its performance. Surprisingly, we found this method to be very effective for a wide range of realistic scenarios. This led us to carry out a theoretical investigation using a *fixed-degree* random graph model for the input supergraph, where every node on L has recommendations to a random subset of size d nodes in R. This is similar to another model usually termed 'd-out', where vertices from both sides of the bipartition generate d edges [**?**]. Our main result (Theorem 1 in Section 2.1) identifies the range of parameters involving $c, a, |L|, |R|$ where the lazy algorithm is very effective. We also show that this is a $(1 - \frac{1}{e})$- approximation algorithm in expectation, and high probability bounds as well. We then analyzed a *greedy* algorithm that scans the nodes on R, and checks if there are a neighbors from L that have not exhausted their bound of c edges and if so, it uses them to add this node to the solution set. We study this algorithm in Section 2.2, where a simple $\frac{1}{a+1}$-approximation result is followed by a very tight bound for realistic values of the edge density under the usual Erdös-Renyi model [**?**]. This analysis serves to portend the impressive overall performance of greedy in our computational results.

Perfect Recommendation Subgraphs. Next we turn to the question of the conditions under which the extensions of perfect matchings (corresponding to the case $a = 1$) exist in random input supergraphs under the standard Erdös-Renyi model. Since at most c edges are allowed out from each vertex in L, and vertices on the R require degree a, the maximum number of nodes on R that have degree a is at most $\frac{cl}{a}$. When a subgraph achieving this bound exists, we say that there is a perfect (c, a)-recommendation subgraph. We extend prior results on the existence of a perfect matching to characterize the edge probability values above which there exist perfect recommendation subgraphs, by using a novel subset partitioning method for analysis.

New Models for Recommendation Supergraphs. Informed by our empirical experience, we identify the systematic ways in which real data sets diverge from the simple fixed-degree model under which we performed the analysis of the lazy engineer's methods. These lead us to define three new graph generation models for the candidate supergraph that allow for varying density between various categories of nodes on both sides: (i) To model sites where the nodes on both sides of the graph are organized according to the same hierarchy, we propose a *hierarchical tree model*, and extend our analysis: these hierarchies may come from topic ontologies in information sites such as Quora or news sites, or from product classifications in retail sites. (ii) We then introduce a simple *cartesian product model* that partitions both sides into disjoint categories and allows varying edge density across different pairs of categories. (iii) To model the effect of strength of a candidate recommendation, we postulate a *weighted graph model* where all edges in the complete bipartite graph get recommendation weights that are identically distributed; we then examine when subgraphs with L-nodes picking a random set of c edges come close to optimizing the number of nodes in R that have total incoming weight of at least one.

Computational Results. To validate our claims of effectiveness of a lazy engineer empirically, we ran several simulations for a range of values of $k(= \frac{l}{r})$ by varying the value of c in each case. We are able to replicate the qualitative properties of our theoretical results for the random choice algorithm in Section 2.1. We also compare its performance against that of the greedy algorithm and a partition-based algorithm used to prove our results on perfect recommendation graphs.

Our results conclusively show that it pays for the engineer to be a little less lazy than choosing a random set of recommendations, by being greedy in choosing the subgraph. This advantage persists for greedy even in practical scenarios with values of a larger than one, a range where our results on random graph models do not show very strong guarantees.

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. We begin by introducing the fixed-degree random graph model for the input supergraph.

2.1 Fixed Degree Model. In this model, we assume that a bipartite graph $G = (L, R, E)$ is generated probabilistically as follows. Each vertex $v \in L$ uniformly samples a set of d neighbors from R. Subgraph H is now sampled from G by uniformly sampling $c \leq d$ edges incident on each vertex. Let $|L| = l$, $|R| = r$ and $k = l/r$. The following theorem gives a lower bound on the expected solution.

Theorem 1. *Let S be the random variable denoting the number of vertices $v \in R$ such that $\deg_H(v) \geq a$. Then*

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

Proof. Let X_{uv} be the indicator variable of the event that the edge uv ($u \in L$, $v \in R$) is in the subgraph that we picked and set $X_v = \sum_{u \in L} X_{uv}$ so that X_v represents the 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 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

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

□

Theorem 2. *The above sampling algorithm gives a $\left(1 - \frac{1}{e}\right)$-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 the optimal solution size $OPT \leq r\max(ck, 1)$. Therefore, by simple case analysis the approximation ratio in expectation is at least $(1 - \exp(-ck))/\min(ck, 1) \geq 1 - \frac{1}{e}$ □

For the $(c, 1)$-recommendation subgraph problem 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 $ck = 1$ 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.

For the general (c, a)-recommendation subgraph problem, if $ck > a$, then the problem is easy on average. 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 for $a = 1$ does not extend here. The following table shows how large ck needs to be for the solution to be 95% optimal for different values of a.

We close out this section by showing that the main result that holds in expectation also hold with high probability. While Chernoff bounds are usually stated for independent variables, the variant below holds for any number of pairwise non-positively correlated variables.

Theorem 3. *[?] Let $X_1, \ldots, X_n$ be non-positively correlated variables. If $X = \sum_{i=1}^{n} X_i$, then for any $\delta \geq 0$*

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

Using this we can convert our expectation result to one that holds with high probability.

Theorem 4. *Let S be the random variable denoting the number of vertices $v \in R$ such that $\deg_H(v) \geq 1$. Then $S \leq r(1 - 2\exp(-ck))$ with probability at most $(e/4)^{r(1-\exp(-ck))}$.*

Proof. We can write S as $\sum_{v \in R} 1 - X_v$ where X_v is the indicator variable that denotes that X_v is matched. 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' has an edge from any vertex $u \in N(v) \cap N(v')$. Also note that the expected size of S is $r(1 - \exp(-ck))$ by Theorem 1. Therefore, we can apply Theorem 3 with $\delta = 1$ and obtain the result. □

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.

We analyze the following natural greedy algorithm that builds a (c, a)-recommendation subgraph iteratively as follows. Consider each vertex in R that has not been selected into H in some arbitrary order. If there is some $v \in R$ that has a neighbors in L all of which have degree less than c, add v and these edges to H breaking ties arbitrarily. Figure **??** illustrates a step of the algorithm.

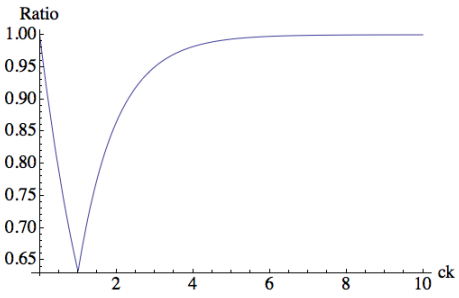

Figure 1: Approx ratio as a function of ck

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 for (c, a)-recommendation subgraph

Theorem 5. *The greedy algorithm achieves $1/(a+1)$-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 $\{u_1, \dots u_a\}$ forms a set of candidate edges that can be used by the greedy algorithm. So we can consider R_{OPT} as a candidate pool for R_{GREEDY}. Each selection that the greedy algorithm makes might result in some of the candidates becoming infeasible. But as long as the candidate pool is not depleted, the greedy algorithm can continue. Each time the greedy algorithm selects some vertex $v \in R$ with edges to $\{u_1, \dots, u_a\}$, we remove v from the candidate pool. If any u_i had degree c in the optimal solution, we would also need to remove an arbitrary vertex $v_i \in R$ adjacent to u_i from 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. 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 can not stop before it has added $OPT/(a+1)$ vertices to the solution. □

Note that this approximation guarantee is as good as we can expect. If we set $a = 1$ then we obtain the familiar 1/2-approximation of the greedy algorithm for matchings. As in matchings, randomizing the order in which the vertices are processed still leaves a constant factor gap in the size of the solution [**?**]. Despite this result, the greedy algorithm fares much better when we give it the same expectation treatment we have given the sampling algorithm. Switching to the Erdös-Renyi model instead of the fixed degree model used in the previous section, we now prove the near optimality of the greedy algorithm for the (c,a)-recommendation subgraph problem.

Theorem 6. *Let $G = (L, R, E)$ be a graph drawn from the $G_{l,r,p}$. If S is the size of the (c,a)-recommendation subgraph produced by the greedy algorithm, then:*

$$E[S] \geq r - \frac{a(lp)^{a-1}}{p}$$

Proof. Note that if edges are generated uniformly, 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 $X_1, \dots, X_i$, the worst condition for X_{i+1} is when all the previous i vertices were from 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 fewer than a of the at least $l - \lceil i/c \rceil$ available vertices have an edge to this vertex is at most $\Pr[Y \sim Bin(l - \frac{i}{c}, p) : Y < a]$. We can bound this probability by bounding each term in the binomial expansion of it by $(1-p)^{l-\frac{ia}{c}-a+1}(lp)^{a-1}$ and obtain:

$$\Pr[Y \sim Bin(l - \frac{ia}{c}, p) : Y < a] \leq a(1-p)^{l-\frac{ia}{c}-a+1}(lp)^{a-1}$$

Summing over all the X_i using the linearity of expectation and this upper bound, we obtain

$$\begin{aligned}
\mathrm{E}[S] &\geq r - \sum_{i=0}^{r-1} \mathrm{E}[\neg X_i] \geq r - \sum_{i=0}^{r-1} \Pr[Y \sim Bin(l - \frac{ia}{c}, p) : Y < a] \\
&\geq r - \sum_{i=0}^{r-1} a(1-p)^{l-\frac{ia}{c}-a+1}[lp - \frac{iap}{c}]^{a-1} \\
&\geq r - a(lp)^{r-1} \sum_{i=0}^{r-1} (1-p)^{l-\frac{ia}{c}-a+1} \geq r - \frac{a(lp)^{a-1}}{p}
\end{aligned}$$

Note that at least asymptotically, this result can explain why the greedy algorithm does much better in expectation than $1/(a+1)$ guarantee we can prove in the worst case. In particular, for a fixed a and c, if l/r is taken to be a constant then this result shows that p only needs to be taken $\Theta(\log^a(l))$ for the greedy algorithm to give $o(r)$ error in expectation.

2.3 Existance of Perfect Recommendation Subgraphs. We now prove existence of perfect recommendation subgraphs which is a generalization of perfect matchings. In the following subsection, We use this to develop our last algorithm. We define a *perfect* (c,a)-recommendation subgraph 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 $\min(r, cl/a)$ of the vertices in R. In this section we will prove a sufficient condition for perfect (c,a)-recommendation subgraphs to exist in a bipartite graph G under the Erdös-Renyi model [**?**] where edges are sampled uniformly and independently with probability p. Our result relies on the following characterization of perfect matchings.

Theorem 7. *[?] Let G be a bipartite graph drawn from $G_{n,n,p}$. If $p \geq \frac{\log n - \log\log n}{n}$, then as $\lim_{n\to\infty}$ probability that G has a perfect matching approaches 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 each of which must exist with high probability if p is sufficiently high. In the next theorem, we show that p only needs to be $\Theta(\frac{\log n}{n})$ for this to succeed. While the theorem is stated for the case when $a \leq c$, it applies equally well to the $a > c$ case by partitioning L instead of R in the following proof.

Theorem 8. *Let G be a random graph drawn from $G_{l,r,p}$ with $p \geq a\frac{\log l - \log\log l}{l}$ and $kc \geq a$, then the probability that G has a perfect (c,a)-recommendation subgraph tends to 1 as $l, r \to \infty$.*

Proof. Given the size and the degree constraints of L, at most lc/a vertices in R can have degree a in a (c,a)-recommendation subgraph. We therefore restrict R to an arbitrary subset R' of size lc/a. Next, we pick an enumeration of the vertices in $R' = \{v_0, \ldots, v_{lc/a-1}\}$ and add these vertices into a subsets by defining $R_i = \{v_{(i-1)l/a}, \ldots, v_{(i-1)l/a+l}\}$ for each $1 \leq i \leq c$ where the arithmetic in the indices is done modulo lc/a. Note both L and all of the R_i have size l.

Using these new sets we define the graphs G_i on the bipartitions (L, R_i). Since the sets R_i are intersecting, we cannot define the graphs G_i to be induced subgraphs. However, note that each vertex $v \in R'$ falls into exactly a of these subsets. Therefore, we can uniformly assign each edge in G to one of a graphs among $\{G_1, \ldots, G_c\}$ it can fall into, and make each of those graphs a random graph. In fact, while the different G_i are coupled, taken in isolation we can consider any single G_i to be drawn from the distribution $G_{l,l,p/a}$ since G was drawn from $G_{l,r,p}$. Since $p/a \geq (\log l - \log\log l)/l$ by assumption, we can now say that by Theorem 7, the probability that a particular G_i has no perfect matching is $o(1)$.

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

2.4 Not-So-Lazy Algorithm Using Perfect Matchings. The above result now enables us to design a near linear time algorithm with a $(1-\epsilon)$ approximation guarantee to the (c,a)-recommendation subgraph problem by leveraging combinatorial properties of matchings.

Theorem 9. *Let G be drawn from $G_{l,r,p}$ where $p \geq a\frac{\log l - \log\log l}{l}$. Then there exists an algorithm that can find a $(1-\epsilon)$-approximation in time $O(\frac{|E|c^2}{\epsilon})$ with probability $1 - o(1)$.*

Proof. As with the proof of the previous theorem, we can arbitrarily restrict R to a subset of size lc/a and divide R into sets $R_1, \ldots, R_c$ such that each vertex in R is contained in exactly a 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 $1 - \epsilon/c$

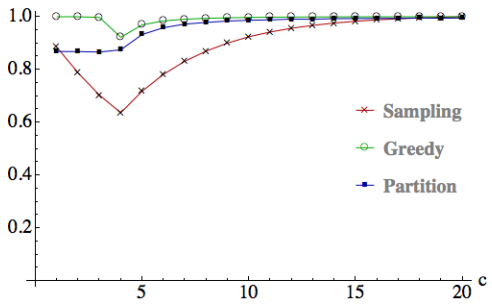

Figure 3: $|L| = 25$k. $|R| = 100$k. $d = 20$. $a = 1$

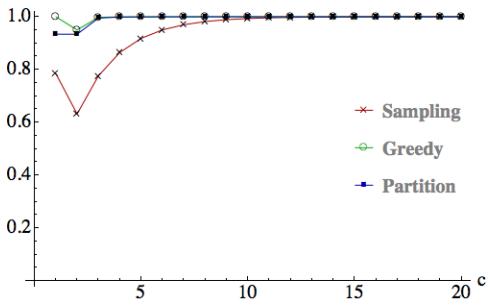

Figure 4: $|L| = 50$k. $|R| = 100$k. $d = 20$. $a = 1$

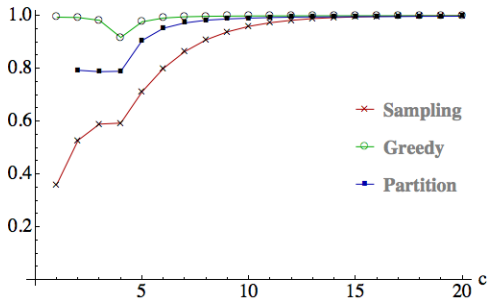

Figure 5: $|L| = 50$k, $|R| = 100$k, $d = 20$, $a = 2$

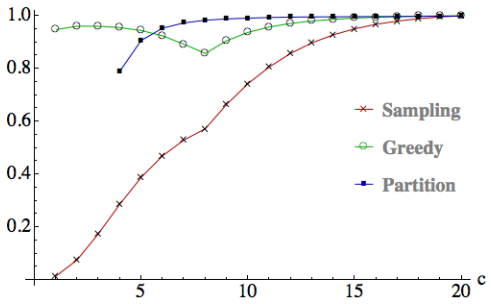

Figure 6: $|L| = 50$k, $|R| = 100$k, $d = 20$, $a = 4$

factor by finding matchings that do not have augmenting paths of length $\geq 2c/\epsilon$ [**?**]. This can be done for each G_i in $O(|E|c/\epsilon)$ time. Furthermore, the union of unmatched vertices makes up an at most $c(\epsilon/c)$ fraction of R'. □

3 Computational Results

We simulated performance of our algorithms on random graphs generated by the graph models we outlined. In the following figures each data point is obtained by averaging the measurements over 10 random graphs. Figures 3 and 4 show that the lower bound we calculated for the expected performance of the sampling algorithm accurately captures the behavior of the sampling algorithm when $a = 1$. Indeed, the inequality we used is an accurate approximation of the expectation, up to lower order terms. The random sampling algorithm does well, both when c is low and high, but falters when $ck = 1$. The greedy algorithm performs better than the random sampling algorithm in all cases, but its advantage vanishes as c gets larger. Note that the dip in the graphs when $cl = ar$, at $c = 4$ in Figure 3 and $c = 2$ in Figure 4 is expected and was previously demonstrated in Figure 1. In all the plots below, the red, blue and yellow lines denote the approximation ratio of the greedy, sampling and partitioning algorithms respectively wrt the upper-bounds of the optimal that we described before.

In contrast to the case when $a = 1$, the sampling algorithm performs worse when $a > 1$ but performs increasingly better with c as demonstrated by Figures 5 and 6. The greedy algorithm continues to produce solutions that are nearly optimal, regardless of the settings of c and a. Therefore, our simulations suggest that in many cases a software engineer can simply design the sampling method for solving the (c, a)-recommendation subgraph problem. In those cases where the sampling is not suitable we still find that the greedy does adequately given its simplicity.

4 Realistic Models of Recommendation Graphs

Even though we studies the fixed-degree model in detail, recommendation systems based on relevance in practice will not have edges that are spread uniformly at random. Items that are

about specific topics are much more likely to interlink within themselves than to those outside that topic, leading to clusters of recommendations.

To understand this substructure in underlying graphs in practice, we compiled results from several e-commerce retailers that have been aggregated and anonymized in the table shown below. For each retailer, we compiled the product ontology present within the site that places a product in this tree-like categorization. E.g., a juicer called "Breville Juice Fountain Plus" is in the tree path: Home → Juicers → High Speed Juicers → Breville Juice Fountain Plus. We then examined the recommendations from products at different depths of the hierarchy. In the table in Figure 7 we examined the edges adjacent to products at depth 4 or greater. We calculated the percentage of edges connecting to products that had different least common ancestors (LCA) with the current product. We then randomized the edges so that we can compare how the graph would have looked if there was no substructure and re-calculate the distribution of the edges and the LCA levels. We noticed that the uniform distribution had edges that had very shallow LCA indicating that most edges did not follow the product hierarchy while in reality, the endpoints of edges recommended had much deeper LCA meaning recommendation edges were clustered based on the product hierarchy. This led us to formalize this new model of input graphs that we study in Subsection 4.1 as the *hierarchical tree model*.

LCALevel	0	1	2	3	4	5	6	7
Uniform	13.4	69.7	12.5	2.6	1.2	0.6	0.0	0.0
Hierarchical	7.1	1.9	8.0	24.9	52.3	5.5	0.2	0.1

Figure 7: Percent edges by LCA of endpoints in reality (Hier) and in simulated uniform distribution.

In a second analysis, we simply truncated the product hierarchy at depth 3 and collected the list of disjoint clusters in the hierarchy. We then examined all the recommendations and partitioned them into those going between each pair of these clusters. In a uniform distribution, we would expect the edges to be equally likely to span across each pair of clusters (assuming clusters are equal sized). But what we observed was that different pairs of clusters had different edge-densities. For instance, an Espresso Machine might point more to other Coffee Machines or Coffee Beans (note that Coffee Beans and Espresso Machine might share no LCA apart from the root) than to other clusters. These results are shown in Figure 8 that clearly demonstrates that these clusters exist and have different densities than the uniform sampling model. This motivated us to define and study the *cartesian product model* in Subsection 4.2 which is orthogonal to the uniform and hierarchical tree models. Finally, in Section 4.3, we study the *weighted model* which assigns weights to the graph edges so that we can incorporate traffic patterns across a website besides just relevance-based recommendations.

4.1 Hierarchical Tree Model. We 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 $|L|/|R| = k$, and require that this ratio is still k if we take any subtree on the left and its corresponding subtree on the right. For simplicity of notations, we will use a subtree and its leaf set interchangeably. We assume that 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 $T_R^{D-2} \backslash T_R^{D-1}$ and so on. The d_i edges out of u are chosen uniformly from $T_R^i \setminus T_R^{i-1}$ and let $d = d_0 + \ldots + d_{D-1}$.

Our goal now is to find a variant of matching [**?**] in this graph that is close to optimal in expectation. That is, our degree upper and lower bounds on vertices in L and R are c and 1 respectively. Let $c = c_0 + \ldots + c_{D-1}$ be similar 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 finds as being generated by the identical graph generation process, but with fewer neighbors selected. With this model and parameters in place, we can have the following analog of our main theorem for $a = 1$ for the hierarchical model and is proved in Appendix A.

Theorem 10. *Let S be the subset of edges $v \in R$ such that $\deg_H(v) \geq 1$. Then*

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

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 by the process corresponding to the lowest common ancestor of u_l and v_r. This way, 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.

4.2 Cartesian Product Model. We can see another shortcoming of the uniform graph model if we think of products as being clustered into categories. This is similar to the hierarchical tree mode, but our assumptions are weaker in that we don't mandate a complete hierarchy and that the clusters can be unrelated. In such a model, we would expect many recommendation edges to be intra-cluster edges rather than inter-cluster edges.

This finding motivates the following model. 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 that is proved in Appendix A.

Theorem 11. *With S, G and H defined as above, we have*

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

An interesting point about this model and the algorithm we described for sampling H is that we are free to select c_{ij}. In particular, c_{ij} can be chosen to maximize the approximation

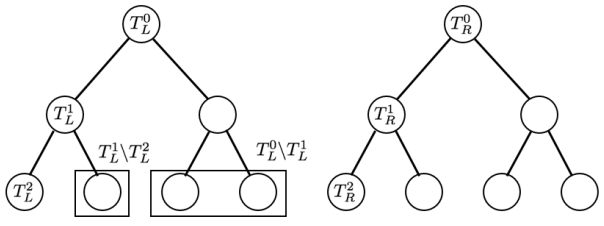

Figure 8: This diagram shows the notation we use for this model and the 1-to-1 correspondance of subtrees.

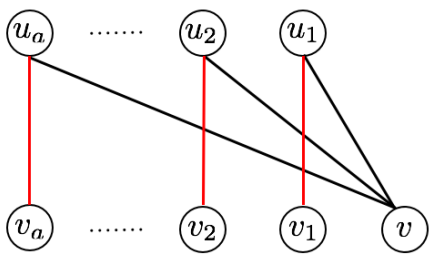

Figure 9: When v selects edges to $u_1, \ldots, u_a$, it can remove $v_1, \ldots, v_a$ from candidates. Potentially invalidated edges shown in red.

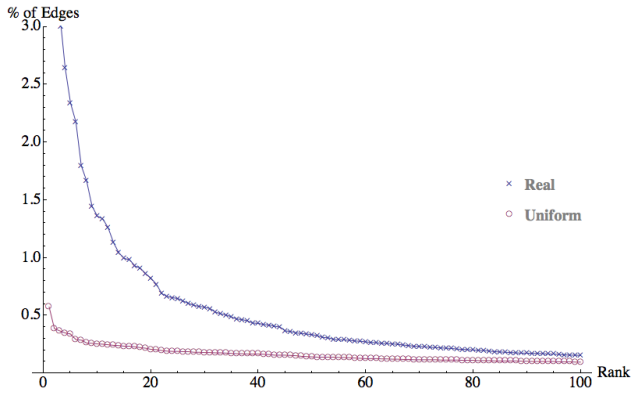

Figure 10: Histogram of percent edges between pairs of clusters. The long tail is omitted.

guarantee in expectation we obtained above using gradient descent or other first order methods prior to running the recommendation algorithm to increases the quality of the solution.

4.3 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 the naive sampling solution we outlined in Section 2.1 still performs exceptionally well. If we let S be the size of the solution produced by this algorithm. We have the following theorem that is proved in Appendix A.

Theorem 12. *Let $G = K_{l,r}$ be a complete bipartite graph where the edges have i.i.d. weights and come from a distribution with mean μ that is supported on $[0, b]$; Assume that $ck\mu \geq 1 + \epsilon$ for some $\epsilon > 0$. If the algorithm from Section 2.1 is used to sample a subgraph H from G, then*

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

There are two things to note about this variant. The first is that since the variables X_v are negatively correlated, our results in Subsection 2.1 can be 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 is sufficient to assume that W_{uv} for all v are independent. In particular, it is possible that W_{uv} for all u are inter-dependent. This allows us to assume a weight distribution that depends on the strength of the recommender and the relevance of the recommendation separately.

5 Conclusions

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 both on the theoretical and the empirical fronts. On the theory side the biggest open problem is the hardness of the general (c, a)-recommendation subgraph problem. For specific values of c and a we know polynomial time algorithms. For e.g., $a = 1$ leads to either bipartite matching or b-matching which can be solved in polynomial time but the hardness of the general problem remains open. While initial experiments are promising, extensive empirical analysis using real data can be revealing in terms of goodness of fit of the models and the performance of the algorithms.

Acknowledgments. The authors would like to thank Alan Frieze for helpful discussions.

References

A Proofs of More General Models

Theorem 10. *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 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 chosen from $r_{D-1}, r_{D-1}, r_{D-2}, \ldots, r_1$ vertices in R as neighbors, 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

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

□

Theorem 11. *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

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

□

Theorem 12. *Let $G = K_{l,r}$ be a complete bipartite graph where the edges have i.i.d. weights and come from a distribution with mean μ that is supported on $[0, b]$; Assume that $ck\mu \geq 1 + \epsilon$ for some $\epsilon > 0$. 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 Hoeffding bounds [**?**] 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

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

□