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 subselect 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 a. We introduce this as the (c, a)-recommendation subgraph problem.

To solve such problems optimally at web-scale graph sizes, implementations would involve 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 simulate traffic patterns. Surprisingly, the lazy engineer would still have found solutions near optimal under different parameters.

*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 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 tday. Seamless site 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 resulting in 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, and retail sites like Amazon that recommend similar products on every product page.

Offline recommendation systems [2, 16, 17] decide on a set of d related candidate items for each item (or page) based on relevance. In this work, we analyze how to prune the set to $c < d$ recommendations such that globally we ensure that the resulting recommendation subgraph is 'efficient' for traversal.

We can represent this notion of recommendations by using a directed graph. A vertex is an item and a directed edge (u, v) is a recommendation from u to v. Note that, 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 [4].

1.1 Bipartite Recommendation Subgraphs

We can generalize the above graph recommendation problem by using a directed bipartite graphs. We can use one partition L to represent the set of items for which we are required to suggest recommendations. We can use the other partition R to represent the set of items that are potentially recommended. A single item can be represented in both L and R if needed. Now the input to the problem is this directed graph G where each vertex has d recommendations. The output required is a subgraph G' where each vertex in L has $c < d$ recommendations. The goal is to minimize the number of vertices that have in-degree less than an integer a. We introduce this as the (c, a)-recommendation subgraph problem. Note that if $a = c = 1$ this is simply the problem of maximum bipartite matching [15]. If $a = 1$ and $c > 1$, we obtain a b-matching problem, that can be converted to the usual bipartite matching problem [9].

Implementation Issues 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 hurdles 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 [1] and hundreds of billions of edges. Even an e-commerce website with 100M product pages and 100 recommendation candidates per product would require easily over 160GB in main memory to run these algorithms without using graph compression techniques that add additional technical difficulties in development and maintenance. In practice offline problem instances are solved by using distributed computing such as map-reduce [5]. However, efficient map-reduce algorithms for graph problems are notoriously difficult and we are back to building a complicated system.

1.2 Algorithms and Analyses

The Lazy Engineer These reasons motivated the authors to investigate the "lazy" approach of choosing a very simple (almost any) set of recommendation to see if they would produce

near optimal solutions at least under realistic scenarios in practice. 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 software designer can consider implementing the classical algorithms.

To be more specific about practical scenarios we briefly describe web graph characteristics to give an idea for the sizes of L, R, c and a to expect. It is well known, and can be verified by prior experiences of the authors that 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 [6, 11]. As demonstrated by one of the prior measurements [14] it is not unreasonable to expect that 50% of the site traffic is contributed by less than 1% (a few thousands) of the web pages and a large number of tail pages (a few hundreds of thousands) contribute the other half of the traffic. 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 work in building these technologies in practice, typical values for c range from 3 to 20 recommendations per page. Values of a are the hardest to nail down but one can think of it as something that can range from 1 to 10.

Recommendation Graph Models The lazy engineer's algorithm essentially picks a random c out of d edges. The natural input graph model to study this is therefore the *fixed-degree* model where every node on the left has recommendations to a random subset of size d nodes in the right. This model is similar to the another model called d-out, where vertices from both sides of the bipartition send d edges to the other side of the bipartition [8]. 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. We extend the models later to more realistic graphs including a hierarchical, cartesian and weighted graph models.

The Greedy Algorithm While the lazy algorithm chooses any set of c recommendations, the greedy algorithm will need some work: it scans the nodes on the right that must be discovered, and checks if there are a neighbors from the left that have not exhausted their bound of c edges in the subgraph, and if so, it uses 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. We use the usual Erdös-Renyi model [7] for the analysis.

Perfect Recommendation Subgraphs Finally we turn to the question of whether the maximum number of nodes allowed by the degree constraints can be covered (at least a times) by a recommendation subgraph: we say that there is an perfect (c, a)-recommendation subgraph in this case. Under the usual Erdös-Renyi model, we use prior results on the existence of a perfect matching to characterize the edge probability over which there exist such perfect recommendation subgraphs, 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.

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 out of the d incident edges 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 $X_v = \sum_{u \in U} X_{uv}$ 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

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

□

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.

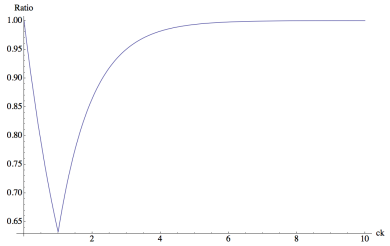

Figure 1: Approximation Ratio as a function of ck

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.

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

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. *[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 $\deg_H(v) \geq a$. 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. 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.

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 $\{u_1, \ldots u_a\}$ 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 $\{u_1, \ldots, u_a\}$, 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. □

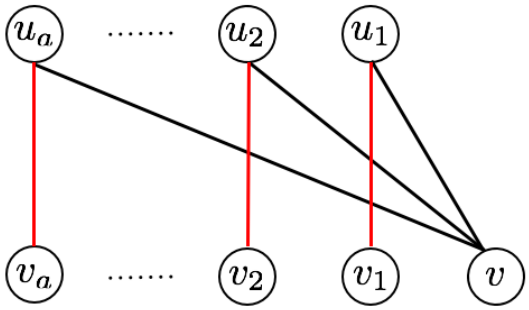

Figure 3: This diagram shows one step of the greedy algorithm. When v selects edges to $u_1, \ldots, u_a$, it potentially removes $v_1, \ldots, v_a$ from the pool of candidates that are avaiable. The potentially invalidated edges are shown in red.

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 [13]. Despite this result, the greedy algorithm fares much better when we give it the same expectation treatment we have given the sampling algorithm. 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 $X_1, \ldots, X_i$, 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 $\Pr[Y \sim Bin(l - \frac{i}{c}, p) : Y < a]$. Using the approximation that

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

and summing over all the X_i using the linearity of expectation, we obtain

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

□

2.3 Existance of Perfect Recommendation Subgraphs

We now prove existance 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 [7] where edges are sampled uniformly and independently with probability p. Our result relies on the following characterization of perfect matchings.

Theorem 7. *[12] 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 R_i and all of the L_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 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 $1 - \epsilon/c$ 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 More Expressive Models of Recommmendation Graphs

We realize that the models described previously are a bit theoretical. In practice, recommendation systems based on relevance will never retrieve edges that are uniformly at random. Items that are about specific topics are much more likely to interlink within themselves than link to those outside of that topic leading to clusters of recommendations rather than a graph where edges are uniformly sampled.

We specifically wanted to understand this substructure in underlying graphs in practice so that we can model the input more realistically. To do this, we compiled results from several ecommerce retailers that have been aggregated and anonymized in the data shown below. For each retailer, we compiled roughly the product ontology present within the site that can place a product in this tree-like hierarchy. For e.g., a juicer called "Breville Juice Fountain Plus" might be in the tree 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 4 we examined the edges adjacent to products at depth 4 or greater. We calculated the percentage of edges connecting to products at different least common ancestors (LCA). 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 meaning the edges did not share much with the product hierarchy while the reality, the edges recommended were much deeper meaning recommendations for say High Speed Juicers were other High Speed Juicers. This immediately led us to formalize this notion of input graphs and we study them in Subsection 3.1 as the *hierarchical tree model*.

Similar to the above analysis, we also trucated the product hierarchy at depth 3 and collected the list of clusters in the hierarchy. We then examined all the recommendations and partitioned

$LCALevel$	0	1	2	3	4	5	6	7	8
Uniform	13.4	69.7	12.5	2.6	1.2	0.6	0.0	0.0	0.0
Hierarchical	7.1	1.9	8.0	24.9	52.3	5.5	0.2	0.1	0.0

Figure 4: Percentage of edges at different LCA levels from the product ontologies of different merchants and as generated by the uniform model

into each pair of these clusters. Again 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. That is, say 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 home) than to other clusters. These results are shown in Figure **??** that clearly demonstrates that these clusters exist and are different from the uniform sampling model. This motivated us to study the *cartesian product model* in Subsection 3.2 which is orthogonal to uniform and hierarchical tree models.

Finally, in Section 3.3 we study the *weighted model* which assigns weights to the graph so that we can incorporate traffic patterns across a website besides just relevance based recommendations.

3.1 Hierarchical Tree Model

While the models where edges are generated uniformly are easy to analyze, they don't accurately capture recommendation graphs. In particular, the set of products a merchant sells can often be captured in a hierarchy of categories. Products that have a low common ancestor are therefore can be very differentiated from the rest of the products and as such much more likely to be able to recommend one another and much less likely to recommend other products in far away branches of this tree. This intuition is validated by an analysis of the category hierarchies of different merchants. The following chart shows products a lot more of the possible recommendations between products are between products that have their lowest common ancestor category at lower levels than the uniform model would generate. This motivates the graph model we describe below.

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 $|L|/|R| = k$, and assume that this ratio still holds 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 $T_R^{D-1} \backslash T_R^{D-2}$ and so on. Each vertex is chosen with uniform probability and we let $d = d_0 + \ldots + d_{D-1}$.

Our goal now is to find a b-matching [9] in this graph that is close to optimal in expectation. That is, our degree upper and bounds on vertices in L is c and 1 respectively. Let $c = c_0 + \ldots + c_{D-1}$ 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 graph generation process, but with fewer neighbors selected. With this model and parameters in place, we can prove the following theorem.

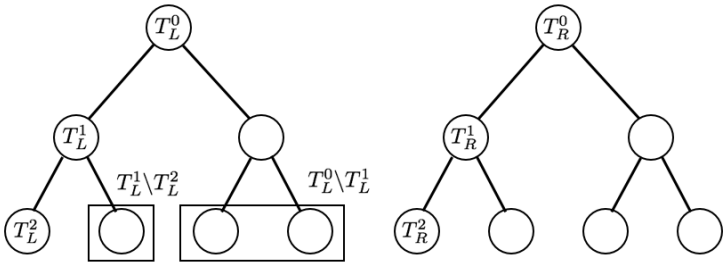

Figure 5: This diagram shows the notation we use for this model and the 1-to-1 correspondance of subtrees

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.

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 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. Assuming that cluster sizes are the same, the uniform model would generate the same number of edges between any two pairs of clusters. However, in an aggregate of data collected from real merchants, we can see that some pairs of clusters have many more edges between them than the uniform model would predict:

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

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

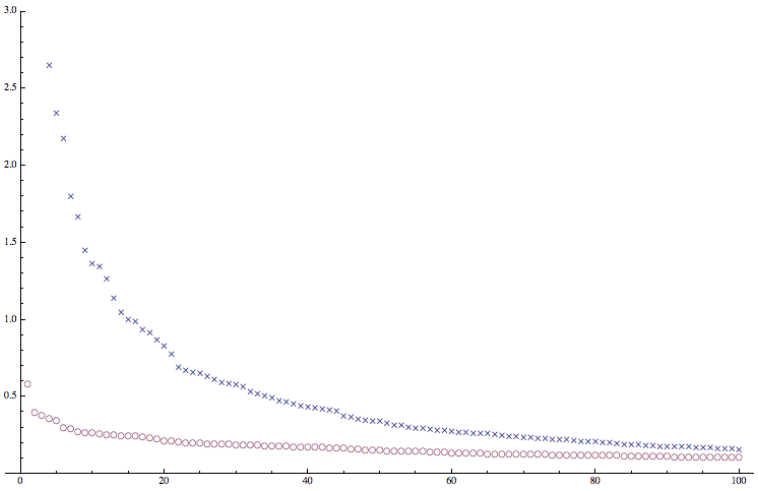

Figure 6: Histogram showing the percentage edges between pairs of clusters. The long tail is omitted.

where the expectation is over G and H.

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. We 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 other first order methods 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 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:

Theorem 12. *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 Hoeffding bounds [10] 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 **??** 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 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 an weight distribution that depends on the strength of the recommender and the relevance of the recommendation separately.

4 Computational Results

We simulated performance of our algorithms on random graphs generated by the graph models we outlined. The following graphs 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 true 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 c gets larger. In all the plots below, the red, blue and yellow lines denote the approximation ratio of the greedy, sampling and partitioning algorithms respectively.

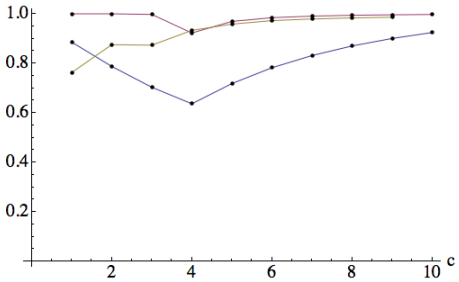

Figure 7: Approximation ratios when $|L| =$ 25k, $|R| = 100$k, $d = 20$, $a = 1$

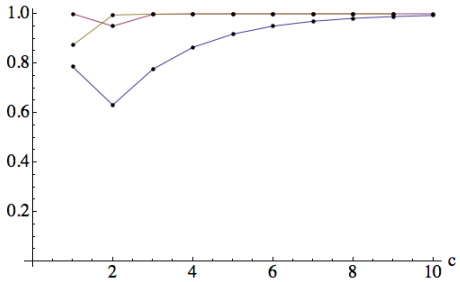

Figure 8: Approximation ratios when $|L| =$ 50k, $|R| = 100$k, $d = 20$, $a = 1$

In contrast to the case when $a = 1$, the sampling algorithm performs much more poorly when $a > 1$. In such cases, the sampling algorithm performs well only when c is large. The greedy algorithm continues to give solutions that are nearly optimal, regardless of the settings of c and a. Therefore, our simulations suggest that in all cases, the greedy algorithm is a suitable and simple replacement for solving the problem to optimality.

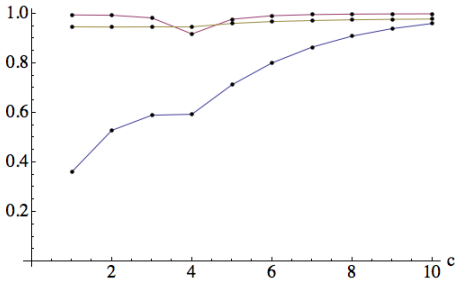

Figure 9: Approximation ratios when $|L| = 50\text{k}$, $|R| = 100\text{k}$, $d = 20$, $a = 2$

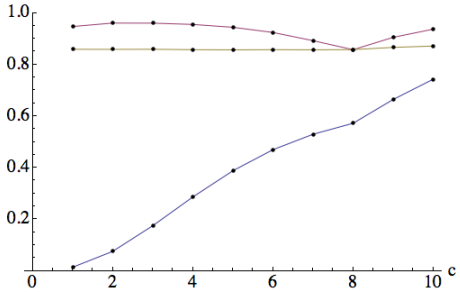

Figure 10: Approximation ratios when $|L| = 50\text{k}$, $|R| = 100\text{k}$, $d = 20$, $a = 4$

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 fractional matching which can be solved in polynomial time but the hardness of the general problem remains open. On the empiral side, the graph models that we introduced can be made richer. For example, the hierarchical graphs require the trees to be balanced and of the same size which is constraining. The edge weights on the weighted graph is sampled from a distribution as opposed to being arbitrary. We expect the research community and our future work to follow-up along both directions in the future.

6 Acknowledgments

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

References

[1] Facebook key facts. urlhttp://newsroom.fb.com/Key-Facts. Accessed: 2013-07-06.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.

[3] Anne Auger and Benjamin Doerr. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. Series on Theoretical Computer Science. World Scientific Publishing Company, 2011.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L Rivest, and Clifford Stein. MIT press, 2001.

[5] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI04: Proceedings of the sixth conference on symposium on operating systems design and implementation*. USENIX Association, 2004.

[6] Bowei Du, Michael Demmer, and Eric Brewer. Analysis of www traffic in cambodia and ghana. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 771–780. ACM, 2006.

[7] P. Erdös and A. Rényi. On random graphs, I. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

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

[9] Harold Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 448–456. ACM, 1983.

[10] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[11] Bernardo A Huberman and Lada A Adamic. Internet: growth dynamics of the world-wide web. *Nature*, 401(6749):131–131, 1999.

[12] Svante Janson, Tomasz Luczak, and Andrzej Rucinski. *Random Graphs*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2011.

[13] Richard Karp, Umesh Vazirani, and Vijay Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358. ACM, 1990.

[14] Chetan Kumar, John B Norris, and Yi Sun. Location and time do matter: A long tail study of website requests. *Decision Support Systems*, 47(4):500–507, 2009.

[15] L. Lovász and M.D. Plummer. *Matching theory*. North-Holland mathematics studies. Akadémiai Kiadó, 1986.

[16] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[17] J. Ben Schafer, Joseph Konstan, and John Riedi. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, EC '99, pages 158–166. ACM, 1999.

A Proofs of More General Models

Theorem 13. *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 14. *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

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

□