Structural Optimization of Recommendations as a Subgraph Selection Problem*

Arda Antikacioglu †
Department of Mathematics
Carnegie Mellon University
aantikac@andrew.cmu.edu

R. Ravi ‡
Tepper School of Business
Carnegie Mellon University
ravi@cmu.edu

Srinath Sridhar
BloomReach Inc.
srinath@bloomreach.com

ABSTRACT

Recommendations are central to the utility of many websites including YouTube, Quora as well as popular e-commerce stores. Such sites typically contain a set of recommendations on every product page that enables visitors to easily navigate the website. Choosing an appropriate set of recommendations at each page is one of the key features of backend engines that have been deployed at several e-commerce sites. Specifically at BloomReach, an engine consisting of several independent components analyzes and optimizes its clients websites. This paper focuses on the structure optimizer component which improves the website navigation experience that enables the discovery of previously undiscovered content.

We begin the paper by formalizing the concept of recommendations used for discovery. We formulate this as a natural graph optimization problem which in its simplest case, reduces to a bipartite matching problem. In practice, solving these matching problems requires superlinear time and is not scalable. Also, implementing simple algorithms is critical in practice because they are significantly easier to maintain in a production software package. This motivated us to analyze three methods for solving the problem in increasing order of sophistication: a local random sampling algorithm, a greedy algorithm and a more involved partitioning based algorithm.

We first theoretically analyze the performance of these three methods on random graph models characterizing when each method will yield a solution of sufficient quality and the parameter ranges when more sophistication is needed. We complement this by providing an empirical analysis of these algorithms on simulated and real-world production data. Our results confirm that it is not always necessary to implement complicated algorithms in the real-world. Indeed, our results demonstrate that very good practical results can be obtained by using simple heuristics that are backed by the confidence of concrete theoretical guarantees.

1. INTRODUCTION

1.1 Web Relevance Engines

The digital discovery divide [13] refers the problem of companies not being able to present users with what they seek in the short time they spend looking for this information. The problem is prevalent not only in e-commerce websites but also in social networks and micro-blogging sites where surfacing relevant content quickly is important for user engagement.

BloomReach is a big-data marketing company that uses the client's content as well as web-wide data to optimize both customer acquisition and satisfaction for e-retailers. BloomReach's clients include popular retailers like Neiman Marcus, Crate & Barrel, Williams-Sonoma and Staples besides many others. In this paper, we describe the structure optimizer component of BloomReach's Web Relevance Engine. This component works on top of the recommendation engine so as to carefully add a set of links across pages that ensures that crawlers as well as users can efficiently navigate the entire website.

1.2 Structure Optimization of Websites

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 [5], it continues to persist even today. Recent estimates [19] attribute up to a third of the sales on Amazon and three-quarters of new orders on Netflix to users that are influenced by the carefully chosen recommendations provided to them.

Even though recommendations exist across the entire www, we provide some simple concrete examples. First, YouTube has a section that displays all the related videos for every main video being viewed. Quora has a section for questions related to the main question that is displayed. These recommendations are critical in determining how the traffic across all of YouTube or Quora is going to flow. An important concern of website owners is whether a significant fraction of the site is not recommended at all (or 'hardly' recommended) from other more popular pages. Continuing with the above example, if a large fraction of the YouTube videos were not recommended from any (or few) other videos, then millions of great videos will lie undiscovered. One way to address this problem is to try to ensure that every page will obtain at least a baseline number of visits so that great content does not remain undiscovered, and thus bridge the discovery divide mentioned above.

We use the criterion of discoverability as the objective for the choice of the links to recommend. Consequently, we get a new formulation of the recommendation selection problem that is structural. In particular, we think of commonly visited pages in a site as the already discovered pages, from

*This work was supported in part by an internship at BloomReach Inc.
†Supported in part by NSF CCF-1347308
‡Supported in part by NSF CCF-1347308

which there are a large number of possible recommendations available to related but less visited peripheral pages. The problem of choosing a limited number of pages to recommend at each discovered page can be cast with the objective of maximizing the number of peripheral non-visited pages that are linked. We formulate this as a recommendation subgraph problem, and study practical algorithms for solving these problems in real-life data.

1.3 Recommendation Systems as a Subgraph Selection Problem

Formally, we divide all pages in a site into two groups: the discovered pages and the undiscovered ones. Furthermore, we assume that recommendation systems [1, 20, 21] find a large set of related candidate undiscovered page recommendations for each discovered page using relevance. In this work, we assume d such related candidates are available per page creating a candidate recommendation bipartite graph (with degree d at each discovered page node). Our goal is to analyze how to prune this set to $c < d$ recommendations such that globally we ensure that the resulting recommendation subgraph can be navigated efficiently by the user to enable better discovery.

1.4 Our Contributions

While optimal solutions to some versions of the recommendation subgraph problem can be obtained by using a maximum matching algorithm, such algorithms are too costly to run on real-life instances. We introduce three simple alternate methods that can be implemented in linear or near-linear time and examine their properties. In particular, we delineate when each method will work effectively on popular random graph models, and when a practitioner needs will need to employ a more sophisticated algorithm. We then evaluate how these simple methods perform on simulated data, both in terms of solution quality and running time. Finally, we show the deployment of these methods on BloomReach's real-world client link graph and measure their actual performance in terms of running-times, memory usage and accuracy.
To summarize, our contributions are as follows.

1. The development of a new structural model for recommendation systems as a subgraph selection problem for maximizing discoverability,
2. The proposal of three methods with increasing sophistication to solve the problem at scale along with associated theoretical performance guarantee analysis for each method, and
3. An empirical validation of our conclusions with simulated and real-life data.

2. RELATED WORK

Recommendation systems have been studied extensively in literature, espcially since the advent of the web. Most recommendation systems can be broadly separated into two different groups: collaborative filtering systems and content-based recommender systems [2]. Much attention has been focused on the former approach, where either users are clustered by considering the items they've consumed or items are clustered by considering the users that have bought them. Both item-to-item and user-to-user recommendation systems based on collaborative filtering have been adopted by many industry giants such as Twitter [11], Amazon [17] and Google [6].

Content based systems instead look at each item and its intrinsic properties. For example, Pandora has categorical information such as Artist, Genre, Year, Singer, Tempo etc. on each song it indexes. Similarly, Netflix has a lot of categorical data on movies and TV such as Cast, Director, Producers, Release Date, Budget, etc. This categorical data can then be used to recommend new songs that are similar to the songs that a user has liked before. Depending on user feedback, a recommender system can learn which of the categories are more or less important to a user and adjust its recommendations.

A drawback of the first type of system is that is that they require multiple visits by many users so that a taste profile for each user, or a user profile for each item can be built. Similarly, content-based systems also require significant user participation to train the underlying system. These conditions are possible to meet for large commerce or entertainment hubs such as the companies mentioned above, but not very likely for most online retailers that specialize in a just a few areas, but have a long-tail [3] of product offerings.

Because of this constraint, in this paper we focus on a recommender system that typically uses many different algorithms that extract categorical data from item descriptions and uses this data to establish weak links between items (candidate recommendations). In the absence of data that would enable us to choose among these many links, we consider every potential recommendation to be of equal value and focus on the objective of discovery, which has not been studied before. In this way, our work differs from all the previous work on recommendation systems that emphasize on finding recommendations of high relevance and quality rather than on structural navigability of the realized link structure.

3. OUR MODEL

We model the structure optimization of recommendations by using a bipartite digraph, where one partition L represents the set of discovered (crawled or often visited) items for which we are required to suggest recommendations and the other partition R representing the set of undiscovered (uncrawled or not visited) items that can be potentially recommended. If needed, the same item can be represented in both L and R.

3.1 The Recommendation Subgraph Problem

We introduce and study this as the **the (c, a)-recommendation subgraph problem** in this paper: *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.*

Note that if $a = c = 1$ this is simply the maximum bipartite matching problem [18]. If $a = 1$ and $c > 1$, we obtain a b-matching problem, that can be converted to a bipartite matching problem [10].

We now describe typical web graph characteristics by discussing the sizes of L, R, c and a in practice. As noted before, 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 [8, 12]. As demonstrated by a prior measurement [16] 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 at BloomReach, 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 5.

3.2 Practical Requirements

Over the past few years, the authors have implemented complex graph algorithms in production software environments. 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, deploy and most importantly maintain long-term. 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, a e-commerce website of a client of BloomReach with 1M 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 [7]. However, efficient map-reduce algorithms for graph problems are notoriously difficult and complicated.

3.3 Simple Solutions

To circumvent the time and space complexity of implementing optimal graph algorithms for the recommendation subgraph problem, we propose the study of three simple solutions strategies that not only can be shown to scale well in practice but also has good theoretical properties.

- **Sampling:** The first solution is a simple random sampling solution that selects a random subset of c links out of the available d from every page. Note that this solution requires no memory overhead to store these results a-priori and the recommendations can be generated using a random number generator on the fly. While this might seem trivial at first, for sufficient (and often real-world) values of c and a we show that this can be optimal.

- **Greedy:** The second solution we propose is a greedy algorithm that chooses the recommendation links so as to maximize the number of nodes in R that can accumulate a in-links. In particular, we keep track of the number of in-links required for each node in R to reach the target of a and choose the links from each node in L giving preference to adding links to nodes in R that are closer to the target in-degree a.

- **Partition:**The third solution is inspired by a theoretically rigorous method to find optimal subgraphs in sufficiently dense graphs: it partitions the edges into a subsets by random sub-sampling, such that there is a good chance of finding a perfect matching from L to R in each of the subsets. The union of the matchings so found will thus result in most nodes in R achieving the target degree a. We require the number of edges in the underlying graph to be significantly large for this method to work very well; moreover, we need to run a (near-)perfect matching algorithm in each of the subsets which is also a computationally expensive subroutine. Hence, even though this method works very well in dense graphs, it does not scale very well in terms of running time and space.

In the next section, we elaborate on these methods, their running times, implementation details, and theoretical performance guarantees. In the following section, we present our comprehensive empirical evaluations of all three methods, first the results on simulated data and then the results on real data from some clients of BloomReach.

4. ALGORITHMS FOR RECOMMENDATION SUBGRAPHS

4.1 The Sampling Algorithm

We present the sampling algorithm for the (c, a)-recommendation subgraph formally below.

Data: A bipartite graph $G = (L, R, E)$
Result: A (c, a)-recommendation subgraph H
for u *in* L **do**
  $S \leftarrow$ a random sample of c vertices without replacement in $N(u)$;
  for v *in* S **do**
    $H \leftarrow H \cup \{(u, v)\}$;
  end
end
return H;

Algorithm 1: The sampling algorithm

Given a bipartite graph G, the algorithm has runtime complexity of $O(|E|)$ since every edge is considered at most once. The space complexity can be taken to be $O(1)$, since the adjacency representation of G can be assumed to be pre-sorted by the endpoint of each edge in L.

We next introduce a simple random graph model for the supergraph from which we are allowed to choose recommendations and present a bound on its expected performance when the underlying supergraph $G = (L, R, E)$ is chosen probabilistically according to this model.

Fixed Degree Model: Each vertex $v \in L$ uniformly and independently samples a set of d neighbors from R. This model is similar to, but is distinct from the more commonly known Erdös-Renyi model of random graphs. In particular, while the degree of each vertex in L is fixed under this model, concentration bounds can show that the degrees of the vertices in L would have similarly been concentrated around d under appropriate parameter settings. We prove the following theorem about the performance of the Sampling Algorithm.

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

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

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

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

□

We can combine this lower bound with a trivial lower bound to obtain an approximation ratio that holds in expectation.

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. Figure 1 shows the approximation ratio as a function of ck.

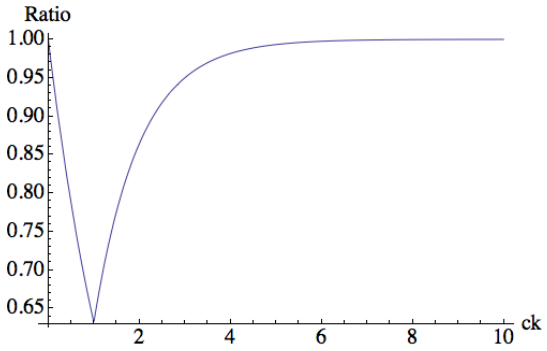

Figure 1: Approx ratio as a function of ck

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

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 table in Figure 2 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 for $a = 1$, using the following variant of Chernoff bounds.

Theorem 3. *[4] Let $X_1, \ldots, X_n$ be non-positively correlated variables. If $X = \sum_{i=1}^n X_i$, then for any $\delta \geq 0$*

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

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 $\delta = 1$ to obtain the result. □

For realistic scenarios where r is very large, this gives very good bounds.

4.2 The Greedy Algorithm

We now analyze the following natural greedy algorithm for constructing a (c, a)-recommendation subgraph H iteratively.

```
Data: A bipartite graph G = (L, R, E)
Result: A (c, a)-recommendation subgraph H
for u in L do
  | d[u] ← 0
end
for v in R do
  | F ← {u ∈ N(v) | d[u] < c};
  | if |F| ≥ a then
  |   | restrict F to a elements;
  |   | for u in F do
  |   |   | H ← H ∪ {(u, v)};
  |   |   | d[u] ← d[u] + 1;
  |   | end
  | end
end
return H;
```

Algorithm 2: The greedy Algorithm

The algorithm loops through each vertex in R, and considers each edge once. Therefore, the runtime is $\Theta(|E|)$. Furthermore, the only data structure we use is an array which keeps track of $\deg_H(u)$ for each $u \in L$, so the memory consumption is $\Theta(|L|)$. Finally, we prove prove the following tight approximation property of this algorithm.

Theorem 5. *The greedy algorithm gives at last a $1/(a+1)$-approximation to 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. Each selection of the greedy algorithm might result in some candidates becoming infeasible, but it can continue as long as the candidate pool is not depleted. 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. Therefore, at each step of the greedy algorithm, we may remove at most $a + 1$ vertices from the candidate pool as illustrated in Figure **??**. Since our candidate pool has size OPT, the greedy algorithm can not stop before it has added $OPT/(a + 1)$ vertices to the solution. □

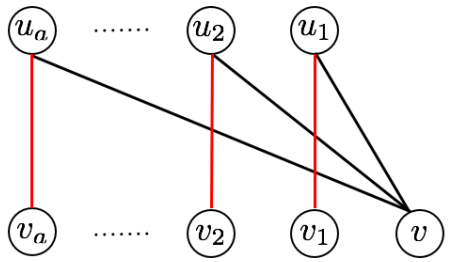

Figure 3: This diagram shows one step of the greedy algorithm. When v selects edges to $u_1, \ldots, u_a$, it potentially removes $v_1, \ldots, v_a$ from the pool of candidates that are available. The potentially invalidated edges are shown in red.

This approximation guarantee is as good as we can expect, since for $a = 1$ we recover 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 quality of the solution [15]. 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:*

$$\mathrm{E}[S] \geq r - \frac{a(lp)^{a-1}}{(1-p)^a} \sum_{i=0}^{r-1} (1-p)^{l - \frac{ia}{c}}$$

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 $\Pr[Y \sim Bin(l - \frac{i}{c}, p) : Y < a]$. We can bound this probability by bounding each term in its binomial expansion by $(1-p)^{l - \frac{ia}{c} - a + 1}(lp)^{a-1}$ to obtain the following.

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

Summing over all the X_i using the linearity of expectation

and this upper bound, we obtain

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

□

Asymptotically, this result explains why the greedy algorithm does much better in expectation than $1/(a+1)$ guarantee we can prove in the worst case. In particular, suppose a and c are fixed and that l/r is taken to be a constant as both l and r tend to ∞. In the realm where sublinear error is possible (i.e. when $lc/a > r$) each term in the sum above becomes $\Theta(l^{-\epsilon})$ for some $\epsilon > 0$ if we set $p = \Theta(\log(l)/l)$. Consequently, the error term reduces to $\Theta(l^{1-\epsilon} \log^a(l))$ which is sublinear on the number of vertices.

4.3 The Partition Algorithm

To motivate the partition algorithm, we first define the idea of optimal solutions for the recommendation subgraph problem.

Perfect Recommendation Subgraphs: 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.

The reason we define perfect (c, a)-recommendation subgraphs is that when one exists, it's possible to recover it in polynomial time using a min-cost b-matching algorithm for any setting of a and c. However, both b-matching algorithms and their implementations often incur significant overheads even over matchings. This motivates a solution that uses only matchings to approximate an optimal solution given that one exists.

We do this by proving a sufficient condition for perfect (c, a)-recommendation subgraphs to exist with high probability in a bipartite graph G under the Erdös-Renyi model [9] where edges are sampled uniformly and independently with probability p. This argument then guides our formulation of a heuristic that overlays matchings in strategic ways to obtain (c, a)-recommendation subgraphs.

Theorem 7. *[14] 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 particular, we show that p only needs to be $\Omega(\frac{\log n}{n})$ for this to succeed.

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 each of these vertices into a subsets as follows. Define $R_i = \{v_{(i-1)l/a}, \ldots, v_{(i-1)l/a+l-1}\}$ 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's 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 randomly 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 conclude 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 then conclude that except for a $o(1)$ probability, each one of the G_i's 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. □

Approximation 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. We call this method the Partition Algorithm, and we outline it below.

Data: A bipartite graph $G = (L, R, E)$
Result: A (c,a)-recommendation subgraph H
$R' \leftarrow$ a random sample of $|L|c/a$ vertices from R;
Choose $G[L, R_1], \ldots, G[L, R_c]$ as in Theorem 8;
for *i in [1..n]* **do**
  $M_i \leftarrow$ A matching of $G[L, R_i]$ with no augmenting path of length $2c/\epsilon$;
end
$H \leftarrow M_1 \bigcup \ldots \bigcup M_c$;
return H;

Algorithm 3: The partition algorithm

Theorem 9. *Let G be drawn from $G_{l,r,p}$ where $p \geq a\frac{\log l - \log\log l}{l}$. Then Algorithm 3 finds a $(1 - \epsilon)$-approximation in $O(\frac{|E|}{\epsilon})$ time with probability $1 - o(1)$.*

Proof. 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 $\geq 2c/\epsilon$ [18]. 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', which proves the claim. □

Notice that if we were to run the augmenting paths algorithm to completeness for each matching M_i, then this algorithm would take $O(|E||L|)$ time. We could reduce this

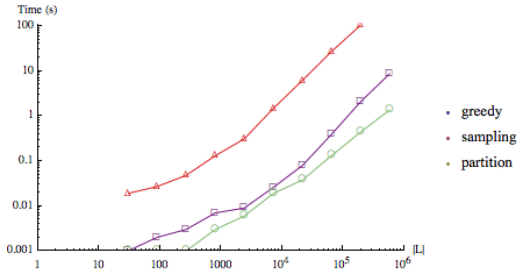

Figure 4: Time needed to solve a (3,3)-recommendation problem in random graphs where $|L|$ scales as $|R|$ with k=4.

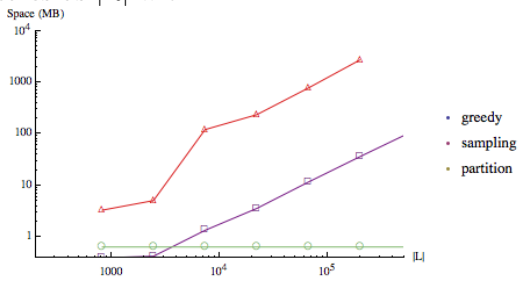

Figure 5: Space needed to solve a (3,3)-recommendation problem in random graphs where $|L|$ scales as $|R|$ with k=4.

further to $O(|E|\sqrt{L})$ by using Hopcroft-Karp. Assuming a sparse graph where $|E| = \Theta(|L|\log|L|)$, the time complexity of running this algorithm comes out to $\Theta(|L|^{3/2}\log|L|)$. The space complexity is only $\Theta(|E|) = \Theta(|L|\log|L|)$, but a large constant is hidden by the big-Oh notation that makes this algorithm impractical in real test cases.

5. EXPERIMENTAL RESULTS

5.1 Simulated Runs

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. We first present the time and space usage of these algorithms when solving a $(3,3)$-recommendation subgraph problem in different sized graphs. Note that varying the value of a and c would only change space and time usage by a constant, so these two graphs are indicative of time and space usage over all ranges of parameters

Recall that the partition algorithm split the graph into multiple graphs and found matchings in these smaller graphs which were then combined into a recommendation subgraph. For this reason, a run of the partition algorithm takes much longer to solve a problem instance than either the sampling or greedy algorithms. It also takes significantly more memory to run. This can easily be seen in **??** and **??**. Compare this to greedy and sampling which both require a single pass over the graph, and no advanced data structures. In fact, if the adjacency list of G is pre-sorted by the edge's endpoint in L, then sampling can be implemented as an online algorithm. Similarly, if the adjacency list of G is pre-sorted by the edge's endpoint in R, then the greedy algorithm can be implemented so that the graph doesn't have to be kept in

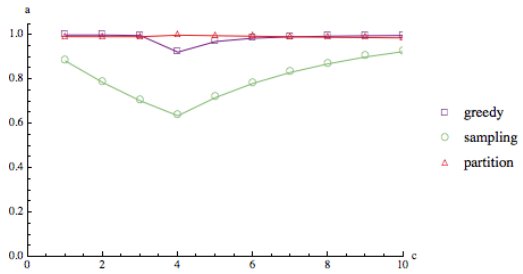

Figure 6: $|L| = 25\mathbf{k}$, $|R| = 100\mathbf{k}$, $d = 20$, $a = 1$

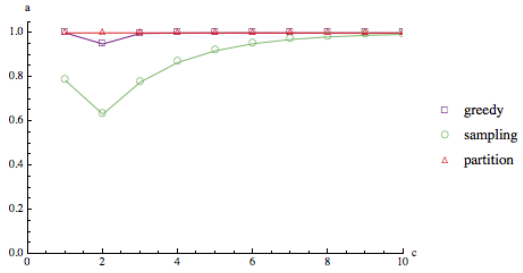

Figure 7: $|L| = 50\mathbf{k}$, $|R| = 100\mathbf{k}$, $d = 20$, $a = 1$

memory. In this event, greedy uses only $O(|L|)$ memory.

Next, we analyze the relative qualities of the solutions each method produces. Figures 6 and 7 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 6 and $c = 2$ in Figure 7 is expected and was previously demonstrated in Figure 1. The partition algorithm is immune to this drop that effects both the greedy and the sampling algorithms.

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 8 and 9. 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 as flagged by our analysis, we still find that the greedy performs adequately and is simple to implement.

In short, our synthetic experiments show the following strengths of each algorithm:

Sampling Algorithm: This algorithm uses little to no memory and can be implemented as an online algorithm. If keeping the underlying graph in memory is an issue, then chances are this algorithm will do well while only needing a fraction of the resources the other two algorithms would take.

Partition Algorithm: This algorithm does well, but only

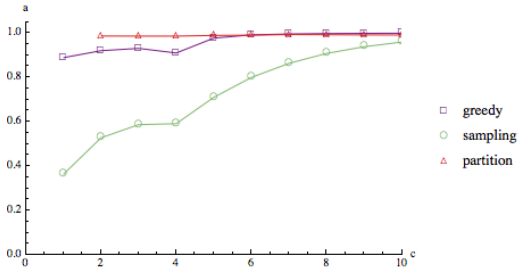

Figure 8: $|L| = 50\mathbf{k}$, $|R| = 100\mathbf{k}$, $d = 20$, $a = 2$

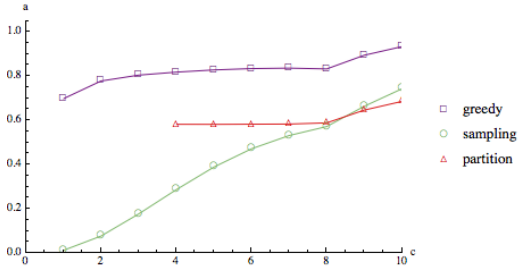

Figure 9: $|L| = 50\mathbf{k}$, $|R| = 100\mathbf{k}$, $d = 20$, $a = 4$

when a is small. In this realm, partition is not only competitive with greedy, but it's actually better. However this performance comes at expense of significant runtime and space costs. This algorithm will recover the optimal solution when $a = 1$, and will likely be able to recover a good (c, a)-recommendation subgraph when we know that a perfect one exists. It also has a distinct advantage over both greedy and sampling algorithms when $lc = ra$.

Greedy Algorithm: This algorithm is the all around best algorithm we tested. It's runs really quickly because it only requires a single pass over the data and uses relatively little amounts of space enabling it run completely in memory for graphs with as many as tens of millions of edges. It's not as quick as sampling or accurate as partition when a is small, but it has great performance over all settings of parameters.

5.2 Real Data

Below, we present the results of running the algorithm on several real life datasets. Two of the merchants had moderately sized relation graphs with about 10^5 vetices and 10^6 relations, while the remaining merchants relations between them while Merchants 3, 4 and 5 have on the order of 10^6 products and 10^7 relations between them. The size of the true optimal solution for these recommendation problems was unknown to us, and we estimated this quantity by taking the minimum of $|L|c/a$ and the number of vertices in R of degree at least a. Note that this is an upper bound on OPT, and that it's almost certain that the true OPT is lower than this value. The line tracked by the following graphs is an average of the optimality percentage of our algorithms across all the merchants. Note that we could only run the partition algorithm for the first two merchants due to memory constraints.

From these results, we can see that that greedy performs exceptionally well when c gets even moderately large. For the realistic value of $c = 6$, the greedy algorithm produced a solution that was 85% optimal for all the merchants we tested. For several of the merchants, it in fact performed almost optimally starting from $a = 2$.

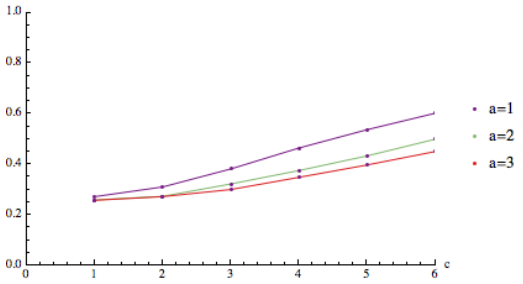

Figure 10: Sampling algorithm for real merchants

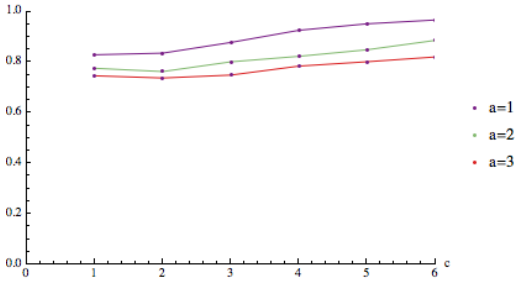

Figure 11: Greedy algorithm for real merchants

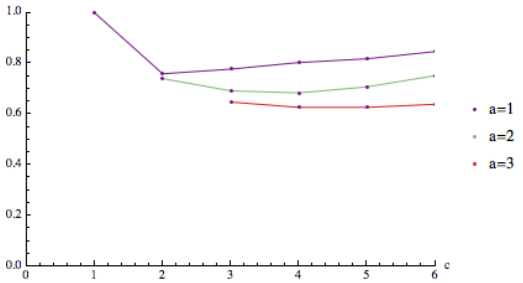

Figure 12: Partition algorithm for real merchants

The partition is also promising, especially when the a value we're aiming for is low. Indeed, when $a = 1$ or $a = 2$, it's on average comparable to greedy though its performance lags behind what the simulated runs would suggest. In some of the low-a instances the partition algorithm can still beat the greedy algorithm. However, as a gets larger, the partition algorithm gets worse faster than the other algorithms since the matchings it finds are not informed of the other matchings.

The sampling algorithm does mostly fine in real life data, but only when c get rather large. It's obviously worse than greedy, but unlike the partition algorithm its performance improves dramatically as c gets larger, and its performance doesn't get worse as quickly when a gets larger. Therefore, as c gets larger, it becomes a viable alternative to greedy especially in cases where we can't even pay the $O(|V|)$ memory cost of the greedy algorithm. It is therefore impractical to use on all but the largest of data sets.

6. REFERENCES

[1] 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.

[2] Dhoha Almazro, Ghadeer Shahatah, Lamia Albdulkarim, Mona Kherees, Romy Martinez, and William Nzoukou. A survey paper on recommender systems. *arXiv preprint arXiv:1006.5278*, 2010.

[3] Chris Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006.

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

[5] Vannevar Bush. As we may think. *Atlantic Monthly*, 176:101–108, 1945.

[6] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDIâĂŹ04: Proceedings of the sixth conference on symposium on operating systems design and implementation*. USENIX Association, 2004.

[8] 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.

[9] Paul Erdös and Albért Rényi. On random graphs, I. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[10] 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.

[11] John Hannon, Mike Bennett, and Barry Smyth. Recommending twitter users to follow using content and collaborative filtering approaches. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 199–206. ACM, 2010.

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

[13] BloomReach Inc. Inside the technology: Web relevance engine.

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

[15] 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.

[16] 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.

[17] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[18] László. Lovász and Michael D. Plummer. *Matching theory*. North-Holland mathematics studies. Akadémiai Kiadó, 1986.

[19] Victor Mayer-Schönberger and Kenneth Cukier. *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. Houghton Mifflin Harcourt, 2013.

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

[21] 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.