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 Sridar
Bloomreach Inc.
srinath@bloomreach.com

1. INTRODUCTION

1.1 Web Relevance Engines

Digital discovery divide (cite Bloomreach white paper) is 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 eCommerce firms but also in social network and micro-blogging sites where surfacing relevant content quickly is important for user engagement.

Bloomreach has a web relevance engine to address the problem that is offered as a service to the websites of popular retailers like Neiman Marcus, Crate & Barrel, Williams-Sonoma and Staples. Main idea is to optimize the site for relevance to the visitors: the site is thoroughly crawled and indexed for semantic interpretation and then eventual relevance scoring. The relevance scoring provides clues for optimizing the navigation structure of the sites that is done by a structure optimizer.

Structure optimizer solves the problem of establishing hyperlinks between the pages in the site for better discoverability. To quote the white-paper, the solution of the structure optimizer is a set of edges that ensure that each "non-crawled" page is easily accessible from a set of "crawled" pages.

1.2 Structure Optimization of Websites

Similar problems arise in all other recommendation systems where the set of possible relevant recommendations from each page is typically much larger than can be displayed as links in the page. This includes e.g., related videos in YouTube, relevant updates in FaceBook feeds, as well as related hashtags in a search for a hashtag in Twitter. All of these sites face the problem of choosing among very relevant content to surface.

If we use the criterion of discoverability as the objective for the choice of the links to recommend, we get a new formulation of the recommendation selection problem that is more structural: In particular, if we think of commonly visited and crawled pages in a site as the core pages, from 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 core page can be cast with the objective of maximizing the number of peripheral non-crawled pages that are linked. We formulate this as a recommendation subgraph problem, and study practical algorithms for solving these problems in real-life data.

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

1.3 Recommendation Systems as a Subgraph Selection Problem

(From previous writeup - needs to be integrated better) 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 [3], it continues to persist even today. The presence of recommendations is an integral feature of several popular websites that are known to have high user engagement and long sessions. Examples range from entertainment sites like YouTube that recommend a set of videos on the right for every video watched by a user, information sites like Quora that recommend a set of related questions on every question page, to retail sites like Amazon that recommend similar products on every product page. Recent estimates [13] attribute up to a third of the sales on Amazon and three-quarters of new orders on Netflix are influenced by precisely choosing recommendations.

Recommendation systems [1, 14, 15] start by finding a large set of related candidate items for each item (or page) using relevance. In this work, we assume d such related candidates are available per page and our goal is to analyze how to prune the set to $c < d$ recommendations such that globally we ensure that the resulting recommendation subgraph can be traversed 'efficiently' by the user.

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. We show how these simpler methods perform very well on simulated data, and have very effective running times.

Two of our simplest methods have been deployed at BloomReach and we show the performance of these methods on data from several retailers that are clients of BloomReach with varying sizes of their sites. Our empirical results clearly show that only the two simplest methods that we study are practically scalable for real-world problems.

To summarize, our contributions are the development of a new structural model for recommendation systems as a

subgraph selection problem for maximizing discoverability, the proposal of very simple methods to solve them at scale along with some associated theoretical performance guarantee analysis, and an empirical validation of our conclusions with simulated and real-life data.

2. RELATED WORK

Recommendation systems surveys and main principles: information retrieval techniques such as PageRank and Cosine similarity for finding related pages.
Work on the classification of pages into core and periphery: why the periphery exists because of the long tail.
Any other work on finding subgraphs in batches and use them to navigate sites? (Any relation to BGP tables in internet routing that get updated dynamically? Those are shortest path tables but here we have the choice of many links to recommend.)

3. OUR MODEL

We model the structure optimization of recommendations by using a bipartite digraph, where one partition L represents the set of crawled items for which we are required to suggest recommendations and the other partition R represents the set of non-crawled 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 **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 [12]. If $a = 1$ and $c > 1$, we obtain a b-matching problem, that can be converted to a bipartite matching problem [7].

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 [5, 8]. As demonstrated by a prior measurement [11] 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 first and third authors have implemented complex graph algorithms at BloomReach. There are two key requirements in making such graph algorithms practical. The first is that the method used must be very simple to implement, debug and deploy. The second is that the method must scale gracefully with larger sizes.

Matching algorithms require linear memory and super-linear run-time which does not scale well. For example, s typical 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 [4]. 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 scale well in practice and have theoretically good performance.

- **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 required no memory overhead to store these results a-priori and the recommendations can be generated using a random number generator on the fly.

- **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**
  neighbors $\leftarrow$ a random sample of c vertices in $N(u)$;
  for v *in neighbors* **do**
    $H \leftarrow H \cup \{(u, v)\}$;
  end
end
return H;

Algorithm 1: The sampling algorithm

Given a bipartite graph G, the algorithm clearly takes linear time since we do $|L|$ iterations of the loop, and a constant amount of work in each iteration. The space complexity is also linear, since the only thing we store is H.

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.

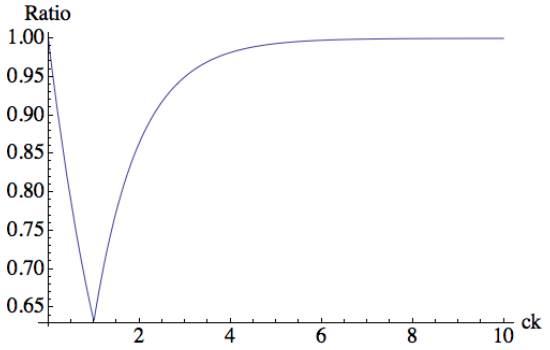

Figure 1: Approx ratio as a function of ck

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. *[2] 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.

The algorithm loops through each vertex in R, and does a constant amount of work in each iteration. Therefore, the runtime is linear. Furthermore, the only data structure we use is an array which keeps track of $\deg_H(u)$ for each $u \in R$, so we only use linear memory as well. 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

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

Algorithm 2: The greedy Algorithm

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. □

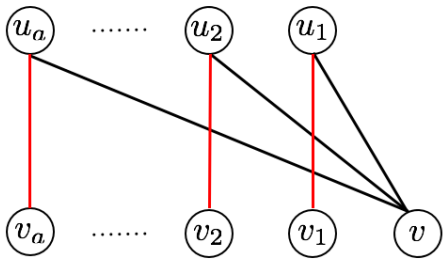

Figure 3: This diagram shows one step of the greedy algorithm. When v selects edges to $u_1, \ldots, u_a$, it potentially removes $v_1, \ldots, v_a$ from the pool of candidates that are available. The potentially invalidated edges are shown in red.

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 [10]. 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.

We will prove a sufficient condition for perfect (c,a)-recommendation subgraphs to exist in a bipartite graph G under the Erdös-Renyi model [6] where edges are sampled uniformly and independently with probability p. We then use the algorithm we propose to prove this condition as the Partition algorithm to compare against random choice and greedy in our tests. Our result relies on the following characterization of perfect matchings.

Theorem 7. *[9] 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.

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 3 finds a $(1-\epsilon)$-approximation in $O(\frac{|E|c^2}{\epsilon})$ 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 $\geq 2c/\epsilon$ [12]. 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. □

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

Depending on the parameters used, the quality of the solutions given by this algorithm can vary wildly. Given the relative difficulty of finding matching in very large scale graphs, we will opt to go for speed more than accuracy in our tests.

5. EXPERIMENTAL RESULTS

******I just copied from the old paper below just to remind ourselves of what plots we need to generate: Needs work.

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, where the red, blue and green lines denote the approximation ratio of the greedy, sampling and partitioning algorithms respectively. 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.

**** Do the figures below only lower bounds, or actual performances as a percentage of the size of $|R|$?

Figures 4 and 5 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 4 and $c = 2$ in Figure 5 is expected and was previously demonstrated in Figure 1.

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 6 and 7. 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.

TODO: Run simulations for much larger sizes of R say 1M, 10M and if possible 100M nodes. Show how far partition can go in these runs and when it is too slow and why.

TODO: Redo the same plots for three different client sets at three different scales: larger R, the better. Again, see if you can run Partition in any of them and when it chokes.

TODO: While running on real data what is the scale for performance that we will be using? Will we just plot the number of nodes of R that had degree at least a in the solution without a benchmark for the size of optimal other than just $|R|$? Can we do anything interesting here?

TODO: Are there other statistics about these three methods that would be illustrative to plot? e.g. running time as we scale up c and the size of R for each of the three programs would be good to measure and plot.

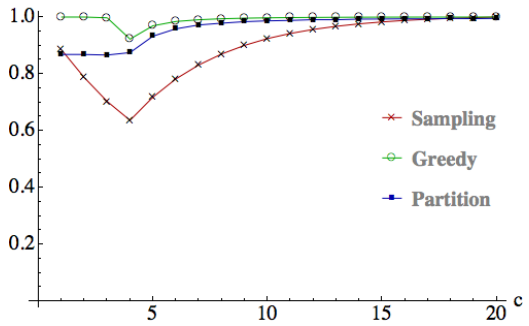

Figure 4: $|L| = 25\mathbf{k}$, $|R| = 100\mathbf{k}$, $d = 20$, $a = 1$

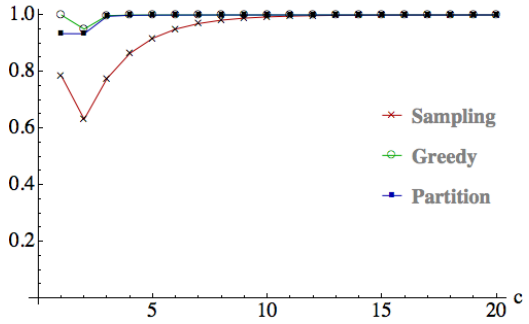

Figure 5: $|L| = 50\mathbf{k}$, $|R| = 100\mathbf{k}$, $d = 20$, $a = 1$

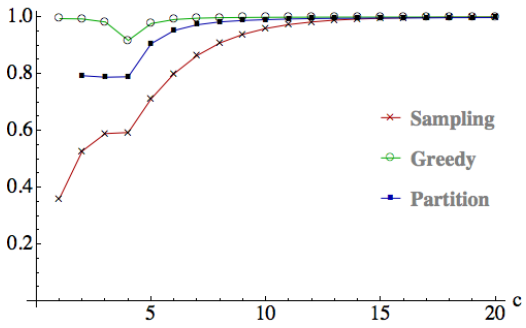

Figure 6: $|L| = 50\mathbf{k}$, $|R| = 100\mathbf{k}$, $d = 20$, $a = 2$

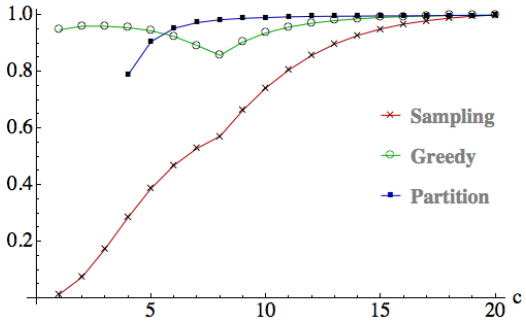

Figure 7: $|L| = 50\mathbf{k}$, $|R| = 100\mathbf{k}$, $d = 20$, $a = 4$

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

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

[4] 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.

[5] 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.

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

[7] 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.

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

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

[10] 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.

[11] 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.

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

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

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

[15] 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.