Recommendation Subgraphs for Web Discovery*

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 popular e-commerce websites. Such sites typically contain a set of recommendations on every product page that enables visitors and crawlers to easily navigate the website. These recommendations are essentially universally present on all e-commerce websites. Choosing an appropriate set of recommendations at each page is a critical task performed by dedicated backend software systems.

We formalize the concept of recommendations used for discovery as a natural graph optimization problem on a bipartite graph: the left partition represent highly visited pages while the right represents rarely visited ones, and the edges are candidate recommendation links that can be used. The goal is to pick at most a fixed number of out-links from each left node to maximize the number of right nodes that are in-linked redundantly. The allowed out-degree of left nodes and minimum redundancy of coverage of right nodes are parameters defining the problem.

We propose 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 and characterize 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 from a retail website. Our results confirm that it is not always necessary to implement complicated algorithms in the real-world, and 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 [15] refers to 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 Nieman 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 users can efficiently navigate the entire website.

1.2 Structure Optimization of Websites

An important concern of retail website owners is whether a significant fraction of the site is not recommended at all (or 'hardly' recommended) from other more popular pages within their site. Moreover, crawlers building search engine indices typically require redundant coverage from several popular pages before they include such less popular pages in their results. One way to address this problem is to try to ensure that every page will obtain at least a baseline number of links from popular pages so that great content does not remain undiscovered, and thus bridge the discovery divide mentioned above. If the website remains connected, this also ensures a simple conductance for the underlying link graph.

We use this criterion of discoverability as the objective for the choice of the links to recommend. We start with a small set of already discovered or popular nodes available at a site, and want to use this set to make as many new nodes discoverable as possible. This objective leads to a new structural formulation of the recommendation selection problem. In particular, we think of commonly visited pages in a site as the already discovered pages, from which there are a large number of possible recommendations available (using more traditional information retrieval methods) 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 redundantly linked. We formulate this as a recommendation subgraph problem, and study practical algorithms for solving these problems at scale with real-life data.

1.3 Recommendation Systems as a Subgraph Selection Problem

Formally, we divide all pages in a site into two groups:

*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

the discovered pages and the undiscovered ones. Furthermore, we assume that traditional recommendation systems [1, 23, 24] provide us with a large set of related candidate undiscovered page recommendations for each discovered page using relevance metrics. 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 number of undiscovered pages that have at least $a \geq 1$ recommendations to them in the chosen subgraph. This gives the (c, a)-recommendation subgraph introduced in Section 3.1. Even though the case of $a = 1$ reduces to a polynomially solvable version of a matching problem, the more usual cases of $a > 1$ are most likely NP-hard prohibiting exact solution methods at scale. Even the simple versions that reduce to matching are too computational expensive on memory and processing to run on real-life instances

1.4 Our Contributions

We introduce three simple heuristic methods that can be implemented in linear or near-linear time and thoroughly investigate their theoretical performance. In particular, we delineate when each method will work effectively on popular random graph models, and when a practitioner 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. It is worthwhile to note that the simplest of the three methods that we propose (sampling) can be easily adapted to the incremental dynamic setting when the set of pages and candidate recommendations is changing rapidly.

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 (Section 3).

2. The proposal of three methods (sampling, greedy and partition) with increasing sophistication to solve the problem at scale along with associated theoretical performance guarantee analyses (Section 4). In particular, we show very strong theoretical bounds on the size of the discoverable set for the sampling algorithm in the fixed degree random graph model (Theorem 1); in the Erdös-Renyi model for the greedy algorithm (Theorem 7) and for a partition-based algorithm (Theorem 10).

3. An empirical validation of our conclusions with simulated and real-life data (Section 5). Our simulations show that sampling is the least resource intensive and performs satisfactorily, while partition is the most resource intensive but performs better for small values of discoverability threshold a; Greedy is the overall best-performer using a single pass over the data and producing good results over a variety of parameters. In the tests with real retailer data, we see these trends broadly reflected in the results: Greedy performs well when c gets moderately large giving almost optimal starting from $a = 2$. The partition method is promising when the targeted a value is low. Sampling is typically worse than greedy, but unlike the partition algorithm, its performance improves dramatically as c becomes larger, and does not worsen as quickly when a gets larger.

2. RELATED WORK

Recommendation systems have been studied extensively in the literature, broadly separated into two different streams: 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 have 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 [12], Amazon [20] 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, 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 other 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. However, while it's not included in this paper for brevity, some of our approaches can be extended to the more general case where different recommendations have different weights (See Theorem 5).

On the graph algorithms side, our problem is related to the bipartite matching and more generally, the maximum b-matching problems. There has been considerable work done in this area. In particular, both the weighted matching and b-matching problems have exact polynomial time solutions [11]. Furthermore the matching problem admits a near linear time $(1 - \epsilon)$-approximation algorithm [9], while the weighted b-matching problem admits a 1/2-approximation

algorithm [18]. However, all such algorithms are based on combinatorial properties of matchings and b-matchings, and do not carry over to the more important version of our problem when $a > 1$.

Finally, our problem bears resemblance to some covering problems. For example, the maximum coverage problem asks for the maximum number of elements that can be covered by a fixed number of sets and has a greedy $(1 - 1/e)$-approximation [22]. However, as mentioned earlier, our formulation requires multiple coverage of elements. Furthermore note that the collection of sets that can be used in the redundant coverage are all possible subsets of c out of the d candidate links, and is expressed implicitly in our problem. The currently known theoretical methods for maximum coverage heavily rely on the submodularity of the objective function, which our objective doesn't satisfy. Hence the line of recent work on approximation algorithms for submodular maximization does not apply to our problems.

3. OUR MODEL

We model the structure optimization of recommendations by using a bipartite digraph, where one partition L represents the set of discovered (i.e., often visited) items for which we are required to suggest recommendations and the other partition R representing the set of undiscovered (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 in R that have in-degree at least a target integer a.*

Note that if $a = c = 1$ this is simply the maximum bipartite matching problem [21]. If $a = 1$ and $c > 1$, we obtain a b-matching problem, that can be converted to a bipartite matching problem [11]. The typical and interesting cases when $a > 1$ is most likely NP-hard, ruling out the possibility of eficient exact algorithms.

The requirement of having a minimum number of a in-links from L for a node in R to be counted as discovered might seem unnatural at first sight. However, it is a natural formulation of the requirement of indexing algorithms for search engines where such a redundancy in their visits is needed before adding pages into the search index. This intuition is supported by the fact that the in-link count of a page provides a local approximation to its PageRank [5].

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, 14, 19]. This is supported by our own experience with the 80/20 rule, i.e. 80% of a site's traffic is captured by 20% of the pages. Therefore, the ratio $k = |L|/|R|$ is typically between 1/3 to 1/5, but may be even lower.

From our own work at BloomReach (and by observing recommendations of Quora, Amazon, and YouTube), 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

There are two key requirements in making 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.

Graph matching algorithms require linear memory and super-linear run-time which does not scale well. For example, an 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 store the graph and run exact matching algorithms; this can be reduced by using graph compression techniques but that adds more technical difficulties in development and maintenance. Algorithms that are time intensive can sometimes be sped-up by using distributed computing techniques such as map-reduce [7]. However, efficient map-reduce algorithms for graph problems are notoriously difficult. Finally, all of these methods apply only to the special case of our problem when $a = 1$, leaving open the question of solving the more interesting and typical cases of redundant coverage when $a > 1$.

3.3 Simple Approximation Algorithms

To satisfy these practical requirements, we propose the study of three simple approximate solutions strategies that not only can be shown to scale well in practice but also have good theoretical properties that we demonstrate using approximation ratios.

- **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. Also, this method is very easy to adapt to the case when the underlying graph is dynamic with both nodes and edges changing over time. Furthermore, our approach can be extended to the case where the recommendation edges have weights representing varying strengths of association as is typically provided by the traditional methods that generate candidate recommendation links[1].

- **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. This method bears close resemblance in strategy with

[1]We omit a full description of this result for brevity.

greedy methods used for maximum coverage and its more general submodular maximization variants.

- **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 edge-subsets which is also a computationally expensive subroutine. Hence, even though this method works very well in dense graphs, its resource requirements may not scale well in terms of running time and space.

As a summary, the table below shows the time and space complexity of our different algorithms.

	Sampling	Greedy	Partition
Time	$O(\|E\|)$	$O(\|E\|)$	$O(\|E\|\sqrt{\|V\|})$
Working Space	$O(1)$	$O(V)$	$O(\|E\|)$

Figure 1: Complexities of the different algorithms (assuming constant a and c)

In the next section, we elaborate on these methods, their running times, implementation details, and theoretical performance guarantees. In the section after that, 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: In this model for generating the candidate recommendation graph, each vertex $u \in L$ uniformly and independently samples d neighbors from R with replacement. While this allows each vertex in L to have the same vertex as a neighbor multiple times, in reality $r \gg d$ is so edge repetition is very unlikely. This model is similar to, but is distinct from the more commonly known Erdös-Renyi model of random graphs [16]. In particular, while the degree of each vertex in L is fixed under our model, concentration bounds can show that the degrees of the vertices in L would have similarly been concentrated around d for $p = d/r$ in the Erdös-Renyi model. We prove the following theorem about the performance of the Sampling Algorithm. We denote the ratio of the size of L and R by k, i.e., we define $k = \frac{l}{r}$.

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*

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

To get a quick sense of the very good performance bounds reflected in this theorem, please see Figure 2 that plots the approximation ratio as a function of ck for the $(c, 1)$-recommendation subgraph problem, as well as Figure 3 that shows how large c needs to be (in terms of k) for the solution to be 95% optimal for different values of a, both in the fixed degree model.

Proof. We will analyze the sampling algorithm as if it picks the neighbors of each $u \in L$ with replacement, the same way the fixed-degree model generates G. This variant would obviously waste some edges, and perform worse than the variant which samples neighbors without replacement. This means that any performance guarantee we prove for this variant holds for our original statement of the algorithm as well.

To prove the claim let X_v be the random variable that represents the degree of the vertex $v \in R$ in our chosen subgraph H. 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 $u \in L$. Since there are cl edges in H that can be incident on v, and each of these edges has a $1/r$ probability of being incident on a given vertex in L, we can now calculate that

$$\begin{aligned}\Pr[X_v = i] &= \binom{cl}{i}(1 - \frac{1}{r})^{cl-i}\left(\frac{1}{r}\right)^i \\ &\leq (cl)^i(1 - \frac{1}{r})^{cl-i}\left(\frac{1}{r}\right)^i\end{aligned}$$

Using a union bound, we can combine these inequalities to upper bound the probability that $\deg_H(v) < a$.

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

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 upper 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.*

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 $ck \to \infty$. When $ck = 3$, 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 2 shows the approximation ratio as a function of ck for the $(c,1)$-recommendation subgraph problem in the fixed degree model.

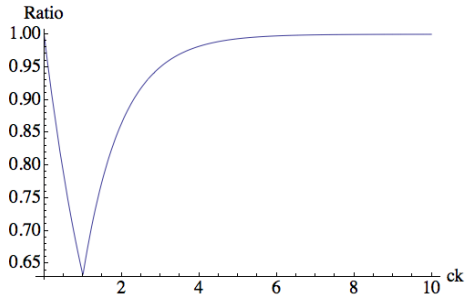


Figure 2: Approx ratio as a function of ck

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, the undiscovered pages in R too numerous to be all covered even if we used the full set of budgeted links allowed out of L, i.e. $cl < ra$ or rearranging, $ck < a$; in this case, we need to use the trivial estimate of ckr/a, and the analysis for $a = 1$ does not extend here. For practical purposes, the table in Figure 3 shows how large c needs to be (in terms of k) for the solution to be 95% optimal for different values of a, again in the fixed degree model.

a	1	2	3	4	5
c	$3.00k^{-1}$	$4.74k^{-1}$	$7.05k^{-1}$	$10.01k^{-1}$	$13.48k^{-1}$

Figure 3: 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 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 $(e/4)^{r(1-\exp(-ck))}$.*

For realistic scenarios where r is very large, the above theorem gives very tight bounds on the size of the solution, also explaining the effectiveness of the simple sampling algorithm in such instances.
The results presented in this section can be naturally extended to weighted models as shown by the theorem below. The proof is left out due to space constraints.

Theorem 5. *Let $G = K_{l,r}$ be a complete bipartite graph where the edges have i.i.d. weights and come from a distribution with mean μ that is supported on $[0,b]$; Assume that $ck\mu \geq 1+\epsilon$ for some $\epsilon > 0$. If the algorithm from Section 4.1 is used to sample a subgraph H from G and S is the set of vertices in R of incident weight at least one, then*

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

4.2 The Greedy Algorithm

We next analyze the natural greedy algorithm for constructing a (c,a)-recommendation subgraph H iteratively. In the following algorithm, we use $N(u)$ to refer to the neighbors of a vertex u.

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 the following tight approximation property of this algorithm.

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

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. Furthermore, even

```
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

in the case of matchings ($a = 1$), randomizing the order in which the vertices are processed is still known to leave a constant factor gap in the quality of the solution [17]. Despite this result, the greedy algorithm fares much better when we analyze its expected performance. Switching to the **Erdös-Renyi model** [10] 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. Recall that in this model (sometimes referred to as $G_{n,p}$), each possible edge is inserted with probability p independent of other edges. In our version $G_{l,r,p}$, we only add edges from L to R each with probability p independent of other edges in this complete bipartite candidate graph. For technical reasons, we need to assume that $lp \geq 1$ in the following theorem. However, this is a very weak assumption since lp is simply the expected degree of a vertex $v \in R$. Typical values for p for our applications will be $\Omega(\log(l)/l)$ making the expected degree $lp = \Omega(\log l)$.

Theorem 7. *Let $G = (L, R, E)$ be a graph drawn from the $G_{l,r,p}$ where $lp \geq 1$. If S is the size of the (c, a)-recommendation subgraph produced by the greedy algorithm, then:*

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

When the underlying random graph is sufficiently dense, Theorem 8 shows that the above guarantee is asymptotically optimal.

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, for a reasonable setting of the right parameters, we can prove that the error term of our greedy approximation will be sublinear.

Theorem 8. *Let $G = (L, R, E)$ be a graph drawn from the $G_{l,r,p}$ where $p = \frac{\gamma \log l}{l}$ for some $\gamma \geq 1$. Suppose that c, a and $\epsilon > 0$ are such that $lc = (1+\epsilon)ra$ and that l and r go to infinity while satisfying this relation. If S is the size of the (c, a)-recommendation subgraph produced by the greedy algorithm, then*

$$E[S] \geq r - o(r)$$

4.3 The Partition Algorithm

To motivate the partition algorithm, we first define 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 $\min(r, \lfloor cl/a \rfloor)$ 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 (matchings with a specified degree b on each vertex) for any setting of a and c. However, implementations of b-matching algorithms often incur significant overheads even over regular bipartite matchings. This motivates a solution that uses regular bipartite matching algorithms to find an approximately optimal solution given that a perfect 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** [10] where edges are sampled uniformly and independently with probability p. This argument then guides our formulation of a heuristic that overlays matchings carefully to obtain (c, a)-recommendation subgraphs.

Theorem 9. *[16] Let G be a bipartite graph drawn from $G_{n,n,p}$. If $p \geq \frac{\log n - \log\log n}{n}$, then as $n \to \infty$, the 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. To find these matchings, we identify subsets of size l in R that we can perfectly match to L. These subsets overlap, and we choose them so that each vertex in R is in a subsets.

Theorem 10. *Let G be a random graph drawn from $G_{l,r,p}$ with $p \geq a\frac{\log l - \log\log l}{l}$ then the probability that G has a perfect (c, a)-recommendation subgraph tends to 1 as $l, r \to \infty$.*

This theorem guarantees the existence of an optimal recommendation subgraph in sufficiently dense subgraphs, and provides a constructive proof of this fact that is also the basis of our partition algorithm.

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. In particular, we use the fact a matching that does not have augmenting paths of length $> 2\alpha$ is a $1 - 1/\alpha$ approximation to the maximum matching problem. We call this method the Partition Algorithm, and we outline it below.

Theorem 11. *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 $\geq 2c/\epsilon$ [21]. 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 10;
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

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 further to $O(|E|\sqrt{L})$ by using Hopcroft-Karp. [13]

Assuming a sparse graph where $|E| = \Theta(|L| \log |L|)$, the time complexity of this algorithm is $\Theta(|L|^{3/2} \log |L|)$. 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 Data

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 100 random graphs. We first present the time and space usage of these algorithms when solving a $(10, 3)$-recommendation subgraph problem in different sized graphs. In all our charts, error bars are present, but too small to be noticeable. 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. The code used conduct these experiments can be found at `https://github.com/srinathsridhar/graph-matching-source`

Recall that the partition algorithm split the graph into multiple graphs and found matchings (using an implementation of Hopcroft-Karp [13]) 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 as can be seen in Figures 5 and 6. Compare this to greedy and sampling which both require a single pass over the graph, and no advanced data structures. In fact, if the edges of G is pre-sorted by the edge's endpoint in L, then the sampling algorithm can be implemented as an online algorithm with constant space and in constant time per link selection. Similarly, if the edges of G is pre-sorted by the edge's endpoint in R, then the greedy algorithm can be implemented so that the entire graph does not have to be kept in memory. In this event, greedy uses only $O(|L|)$ memory.

Next, we analyze the relative qualities of the solutions each method produces. Note that in all of the plots, the purple, red and green lines depict the greedy, partition and sampling algorithms respectively. Plots (a) and (b) of Figure **??** depict the average performance ratio of the three methods

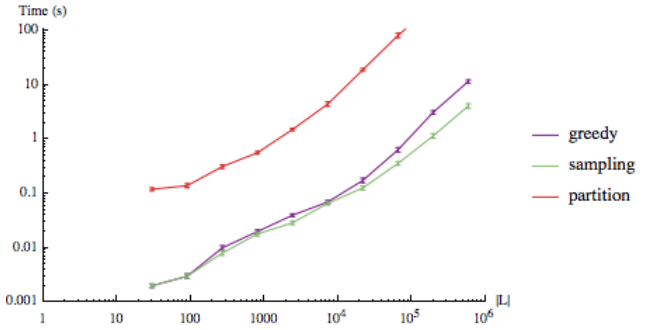


Figure 4: Time needed to solve a (10,3)-recommendation problem in random graphs where $|R|/|L| = 4$ (Notice the log-log scale.)

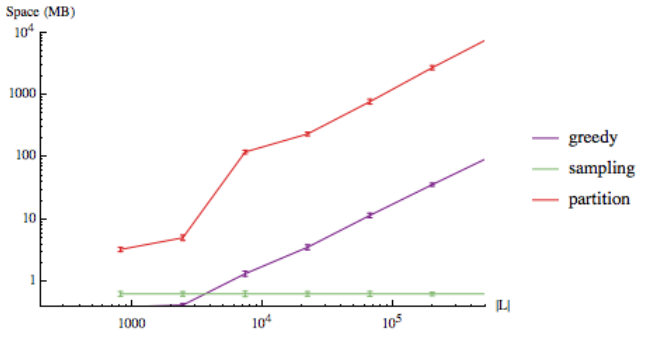


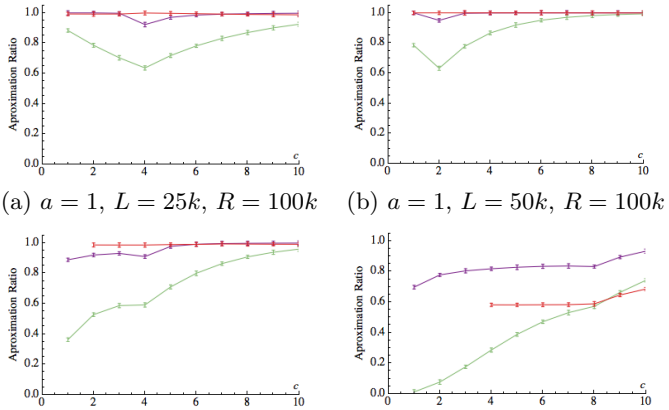
Figure 5: Space needed to solve a (10,3)-recommendation problem in random graphs where $|R|/|L| = 4$ (Notice the log-log scale.)

compared to the trivial upper bounds as the value of c, the number of recommendations allowed is varied, while keeping $a = 1$. They collectively show that the lower bound we calculated for the expected performance of the sampling algorithm accurately captures its behavior when $a = 1$. Indeed, the inequality we used is an accurate approximation of the expectation, up to lower order terms, as is demonstrated in these simulated runs. The random sampling algorithm does well, both when c is low and high, but falters when $ck = 1$. The greedy algorithm outperforms the 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 plot (a) and $c = 2$ in plot (b) of Figure **??** expected and was previously demonstrated in Figure 2. The partition algorithm is immune to this drop that affects both the greedy and the sampling algorithms, but comes with the cost of higher time and space utilization.

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 plots (b) and (d) in Figure **??**. The greedy algorithm continues to produce solutions that are nearly optimal, regardless of the settings of c and a, even beating the partition algorithm with increasing values of a. Our simulations suggest that in most cases, one can simply use our sampling method for solving the (c, a)-recommendation subgraph problem. In cases where the sampling is not suitable as flagged by our analysis, we still find that the greedy performs adequately and is also simple to implement. These two algorithms thus confirm to our requirements we initially laid out for deployment in large-scale real systems in practice.

To summarize, our synthetic experiments show the following strengths of each algorithm:

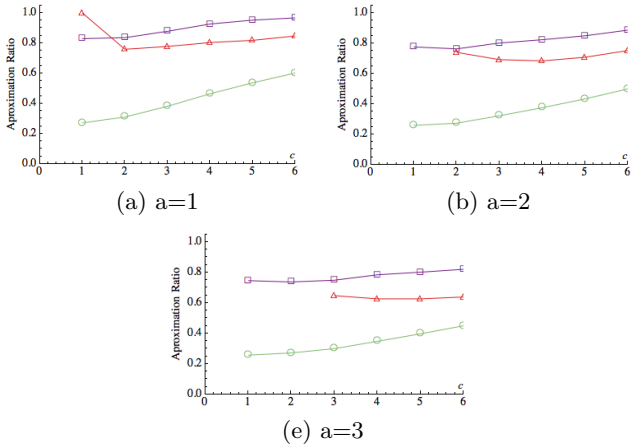
Sampling Algorithm: Sampling uses little to no memory



(a) $a = 1$, $L = 25k$, $R = 100k$ (b) $a = 1$, $L = 50k$, $R = 100k$

(c) $a = 2$, $L = 50k$, $R = 100k$ (d) $a = 4$, $L = 50k$, $R = 100k$

Figure 6: The approximation performance of our algorithms when solving the (c, a)-recommendation subgraph problem. In each plot, c varies along the x-axis and the settings of a and L and R are fixed



(a) a=1 (b) a=2

(e) a=3

Figure 7: Solution quality for the (c, a)-recommendation subgraph problem in retailer data

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 need.

Partition Algorithm: This algorithm does well, but only when a is small. In particular, when $a = 1$ or 2, partition seems to be the best algorithm, but the quality of the solutions degrade quickly after that point. However this performance comes at expense of significant runtime and space. Since greedy performs almost as well without requiring large amounts of space or time, partition is best suited for instances where a is low the quality of the solution is more important than anything else.

Greedy Algorithm: This algorithm is the all-round best performing algorithm we tested. It only requires a single pass over the data thus very quickly, 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 is not as fast as sampling or accurate as partition when a is small, but it has very good performance over all parameter ranges.

5.2 Real Data

We now present the results of running our algorithms on several real datasets. In the graphs that we use, each node corresponds to a single product in the catalog of a merchant and the edges connect similar products. For each product up to 50 most similar products were selected by a proprietary algorithm of BloomReach that uses text-based features such as keywords, color, brand, gender (where applicable) as well as user browsing patterns to determine the similarity between pairs of products. Such algorithms are commonly used in e-commerce websites such as Amazon, Overstock, eBay etc to display the most related products to the user when they are browsing a specific product.

Two of the client merchants of BloomReach presented here had moderate-sized relation graphs with about 10^5 vertices and 10^6 input edges (candidate recommendations); the remaining merchants (3, 4 and 5) have on the order of 10^6 vertices and 10^7 input edges between them. We estimated an upper bound on the optimum solution by taking the minimum of $|L|c/a$ and the number of vertices in R of degree at least a. Figure 7 plots the average of the optimality percentage of the sampling, greedy and partition algorithms across all the merchants respectively. 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, its results were almost optimal starting from $a = 2$.

The partition method is also promising, especially when the a value that is targeted is low. Indeed, when $a = 1$ or $a = 2$, its performance is comparable or better than greedy, though the difference is not as pronounced as it is in the simulations. However, for larger values of a the partition algorithm performs worse.

The sampling algorithm performs mostly well on real data, especially when c is large. It is typically worse than greedy, but unlike the partition algorithm, its performance improves dramatically as c becomes larger, and its performance does not worsen as quickly when a gets larger. Therefore, for large c sampling becomes a viable alternative to greedy mainly in cases where the linear memory cost of the greedy algorithm is too prohibitive.

6. SUMMARY AND FUTURE WORK

We have presented a new class of structural recommendation problems cast as computationally hard subgraph selection problems, and analyzed three algorithmic strategies to solve these problems. The sampling method is most efficient, the greedy approach trades off computational cost with quality, and the partition method is effective for smaller problem sizes. We have proved effective theoretical bounds on the quality of these methods, and also substantiated them with experimental validation both from simulated data and real data from retail web sites. Our findings have been very useful in the deployment of effective structural recommendations in web relevance engines that drive many of the leading websites of popular retailers.

Acknowledgments: We thank Alan Frieze and Ashutosh

Garg for helpful discussions.

7. REFERENCES

[1] G. Adomavicius and A. 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] D. Almazro, G. Shahatah, L. Albdulkarim, M. Kherees, R. Martinez, and W. Nzoukou. A survey paper on recommender systems. *arXiv preprint arXiv:1006.5278*, 2010.

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

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

[5] Ziv Bar-Yossef and Li-Tal Mashiach. Local approximation of pagerank and reverse pagerank. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 279–288. ACM, 2008.

[6] A. S. Das, M. Datar, A. Garg, and S. 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] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI '04: Proceedings of the sixth conference on symposium on operating systems design and implementation*. USENIX Association, 2004.

[8] B. Du, M. Demmer, and E. 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] Ran Duan and Seth Pettie. Approximating maximum weight matching in near-linear time. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 673–682. IEEE, 2010.

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

[11] H. 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.

[12] J. Hannon, M. Bennett, and B. 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.

[13] J. E. Hopcroft and R. M. Karp. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

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

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

[16] S. Janson, T. Luczak, and A. Rucinski. *Random Graphs*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2011.

[17] R. M. Karp, U. Vazirani, and V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. ACM, 1990.

[18] Christos Koufogiannakis and Neal E Young. Distributed fractional packing and maximum weighted b-matching via tail-recursive duality. In *Distributed Computing*, pages 221–238. Springer, 2009.

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

[20] G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

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

[22] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978.

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

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