```
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

Theorem 6. *The greedy algorithm gives 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. Furthermore each u_i could have degree c in the optimal solution and used each of its edges to make a neighbor attain degree a. The greedy choice of an edge to u_i requires us to remove such an edge to an arbitrary vertex $v_i \in R$ adjacent to u_i in the optimal solution, and thus remove v_i from further consideration in the candidate pool. Therefore, at each step of the greedy algorithm, we may remove at most $a + 1$ vertices from the candidate pool as illustrated in Figure 4. Since our candidate pool has size OPT, the greedy algorithm can not stop before it has added $OPT/(a + 1)$ vertices to the solution. □

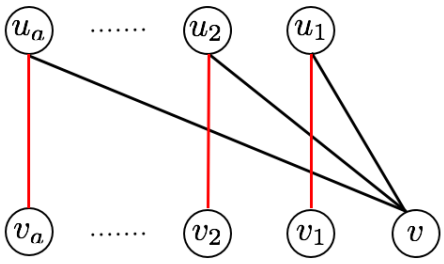

Figure 4: One step of the greedy algorithm. When v selects edges to $u_1, \ldots, u_a$, it can remove $v_1, \ldots, v_a$ from the pool of candidates that are available. The potentially invalidated edges are shown in red.

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. Furthermore, even 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.

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+1} that the greedy algorithm matches the $(i+1)^{st}$ 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 $(i+1)^{st}$ vertex. The maximum number of such invalidated vertices is at most $\lceil ia/c \rceil$. Therefore, the bad event is that we have fewer than a of the at least $l - \lceil ia/c \rceil$ available vertices having an edge to this vertex. The probability of this bad event is at most $\Pr[Y \sim Bin(l - \frac{ia}{c}, p) : Y < a]$, the probability that a Binomial random variable with $l - \frac{ia}{c}$ trials of probability p of success for each trial has less than a successes. We can bound this probability by using a union bound and upper-bounding $\Pr[Y \sim Bin(l - \frac{ia}{c}, p) : Y = t]$ for each $0 \leq t \leq a - 1$. By using the trivial estimate that $\binom{n}{i} \leq n^i$ for all n and i, we obtain:

$$\begin{aligned}
\Pr[Y \sim Bin(l - \frac{ia}{c}, p) : Y = t] &= \binom{l - \frac{ia}{c}}{t} (1-p)^{l-\frac{ia}{c}-t} p^t \\
&\leq \left(l - \frac{ia}{c}\right)^t (1-p)^{l-\frac{ia}{c}-t} p^t \\
&\leq (lp)^t (1-p)^{l-\frac{ia}{c}-t}
\end{aligned}$$

Notice that the largest exponent lp can take within the bounds of our sum is $a-1$. Similarly, the smallest exponent $(1-p)$ can take within the bounds of our sum is $l - \frac{ia}{c} - a + 1$. Now applying the union bound gives:

$$\begin{aligned}
&\Pr[Y \sim Bin(l - \frac{ia}{c}, p) : Y < a] \\
&\leq \sum_{t=0}^{a-1} \Pr[Y \sim Bin(l - \frac{ia}{c}, p) : Y = t] \\
&\leq \sum_{t=0}^{a-1} (lp)^t (1-p)^{l - \frac{ia}{c} - t} \\
&= a(lp)^{a-1}(1-p)^{l - \frac{ia}{c} - a + 1}
\end{aligned}$$

Finally, 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, 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*

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

Proof. We will prove this claim by applying Theorem 7. Note that it suffices to prove that $(lp)^{a-1} \sum_{i=0}^{r-1} (1-p)^{l - \frac{ia}{c}} = o(r)$ since the other terms are just constants. We first bound the elements of this summation. Using the facts that $p = \frac{\gamma \log l}{l}$, $lc/a = (1+\epsilon)r$ and that $i < r$ throughout the summation, we get the following bound on each term:

$$\begin{aligned}
(1-p)^{l - \frac{ia}{c}} &\leq \left(1 - \frac{\gamma \log l}{l}\right)^{l - \frac{ia}{c}} \\
&\leq \exp\left(-\frac{\gamma \log l}{l}\left(l - \frac{ia}{c}\right)\right) \\
&= \exp\left((-\log l)\left(\gamma - \frac{ia}{lc}\right)\right) \\
&= l^{-\gamma + \frac{ia}{lc}} = l^{-\gamma + \frac{i}{(1+\epsilon)r}} \\
&\leq l^{-1 + \frac{1}{1+\epsilon}} = l^{-\frac{\epsilon}{1+\epsilon}}
\end{aligned}$$

Finally, we can evaluate the whole sum:

$$\begin{aligned}
(lp)^{a-1} \sum_{i=0}^{r-1} (1-p)^{l - \frac{ia}{c}} &\leq \left(\log^{a-1} l\right) \sum_{i=0}^{r-1} l^{-\frac{\epsilon}{1+\epsilon}} \\
&\leq \left(\log^{a-1} l\right) r l^{-\frac{\epsilon}{1+\epsilon}} \\
&= \left(\log^{a-1} l\right) \frac{c}{(1+\epsilon)a} l^{1 - \frac{\epsilon}{1+\epsilon}} = o(l)
\end{aligned}$$

However, since r is a constant times l, any function that is $o(l)$ is also $o(r)$ and this proves the claim.

□

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 bipartite 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.

Proof. We start by either padding or restricting R to a set of $\frac{lc}{a}$ before we start our analysis. If $r \geq \frac{lc}{a}$, then we restrict R to an arbitrary subset R' of size $\frac{lc}{a}$. Since induced subgraphs of Erdös-Renyi graphs are also Erdös-Renyi

graphs, we can instead apply our analysis to the induced subgraph. Since the optimal solution has size bounded above by $\frac{lc}{a}$ a perfect (c, a)-recommendation subgraph in $G[L, R']$ will imply a perfect recommendation subgraph in $G[L, R]$.

On the other hand, if $r \leq \frac{lc}{a}$, then we can pad R with $\frac{lc}{a} - r$ dummy vertices and adding an edge from each such vertex to each vertex in L with probability p. We call the resulting right side of the graph R'. Note that $G[L, R']$ is still generated by the Erdös-Renyi process. Further, since the original graph $G[L, R]$ is a subgraph of this new graph, if we prove the existence of a perfect (c, a)-recommendation subgraph in this new graph, it will imply the existence of a perfect recommendation subgraph in $G[L, R]$.

Having picked an R' satisfying $|R'| = \frac{lc}{a}$, 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 9, the probability that a particular G_i has no perfect matching is $o(1)$.

If we fix c, we can conclude by a union bound 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. 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 a bipartite random graph 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..c]* **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. Figures 7 and 8 plot the average performance ratio of the three methods compared to the trivial upper bounds as the value of c, the number of recommendations allowed is varied, while keeping $a = 1$. They col-

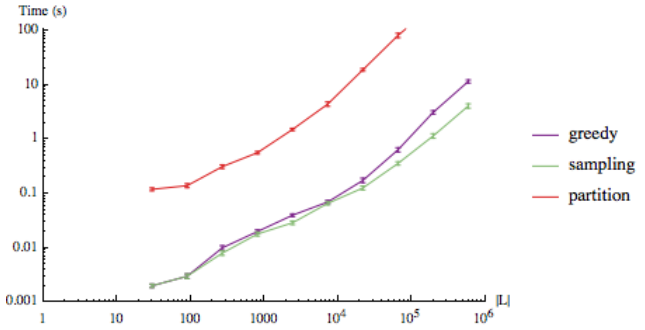

Figure 5: Time needed to solve a (10,3)-recommendation problem in random graphs where $|R|/|L| = 4$ (Notice the log-log scale.)

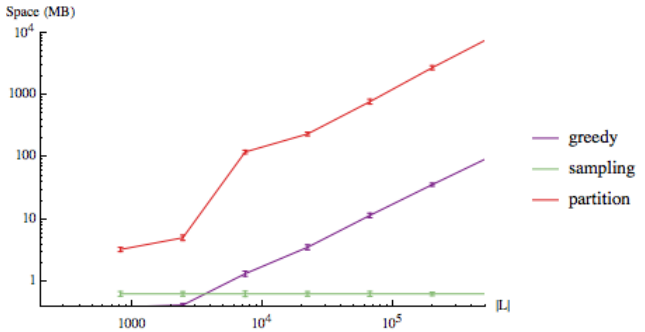

Figure 6: Space needed to solve a (10,3)-recommendation problem in random graphs where $|R|/|L| = 4$ (Notice the log-log scale.)

lectively 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 Figure 7 and $c = 2$ in Figure 8 is 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 Figures 9 and 10. 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 and can be implemented as an online algorithm. If keeping

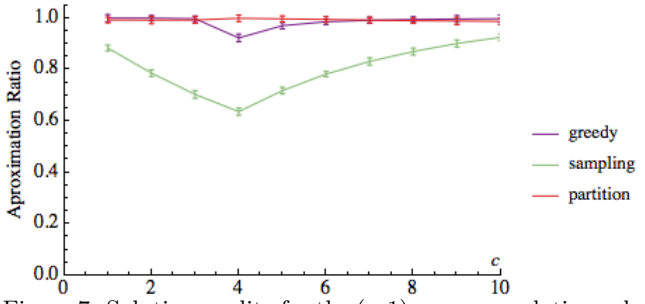

Figure 7: Solution quality for the $(c, 1)$-recommendation subgraph problem in graphs with $|L| = 25$k, $|R| = 100$k, $d = 20$

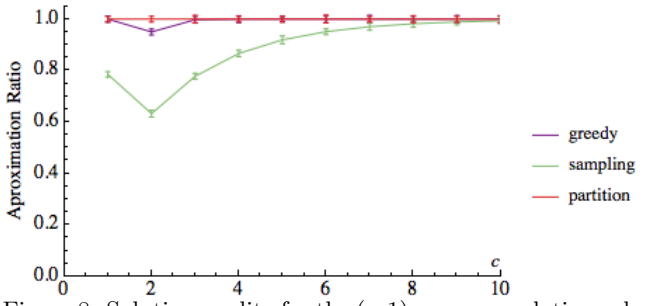

Figure 8: Solution quality for the $(c, 1)$-recommendation subgraph problem in graphs with $|L| = 50$k, $|R| = 100$k, $d = 20$

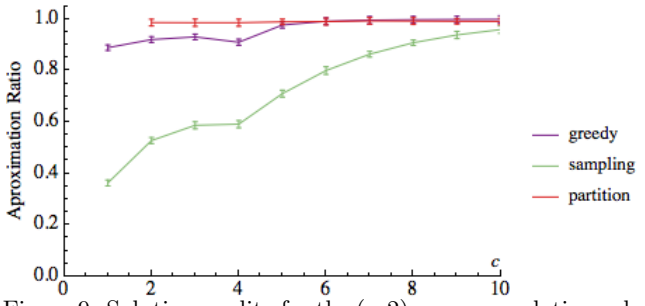

Figure 9: Solution quality for the $(c, 2)$-recommendation subgraph problem in graphs with $|L| = 50$k, $|R| = 100$k, $d = 20$

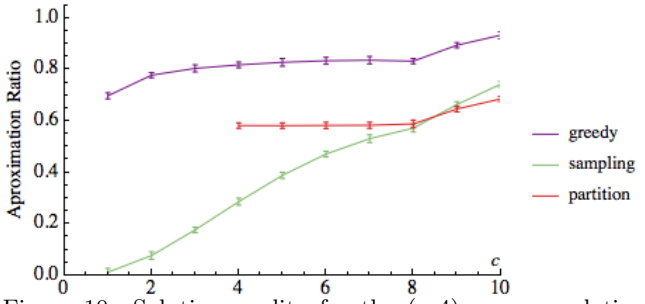

Figure 10: Solution quality for the $(c, 4)$-recommendation subgraph problem in graphs with $|L| = 50$k, $|R| = 100$k, $d = 20$

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. Figures 11, 12 and 13 plot 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.

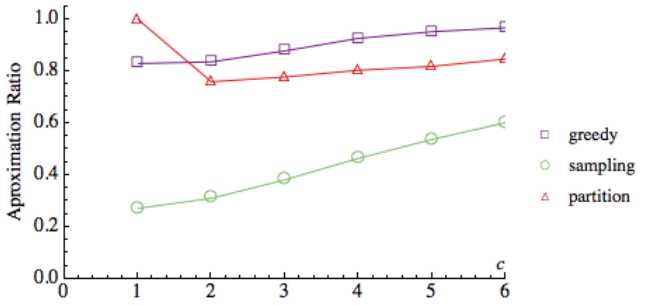

Figure 11: Solution quality for the $(c, 1)$-recommendation subgraph problem in retailer data

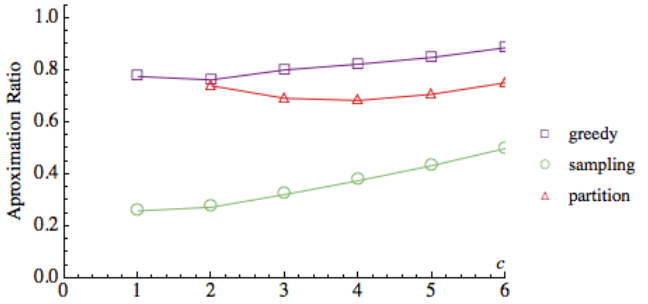

Figure 12: Solution quality for the $(c, 2)$-recommendation subgraph problem in retailer data

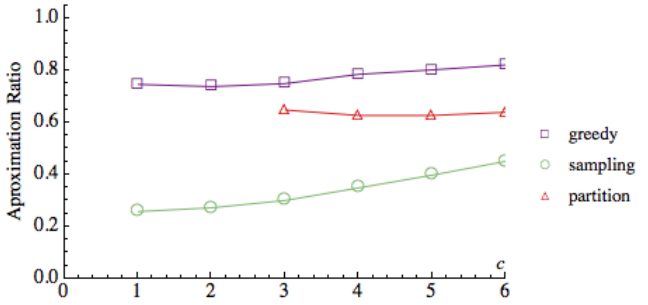

Figure 13: Solution quality for the $(c, 3)$-recommendation subgraph problem in retailer data

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.

We believe that our work lends itself to promising future work in two directions. The first is that through a better

understanding of the underlying graph's topology, more precise or complex models can be used. This would require an empirical validation of the proposed graph model and the adaptation of our methods to different random graph models. We have made some initial progress on this front, which can be found in the full version of the paper.

The second is that most of our algorithms aren't particularly suited for the weighted setting. While our sampling result carries over to the weighted regime as seen in Theorem 5, our other algorithms don't, and even this result is weak compared to the unweighted result we presented in full. The problem presented by ignoring weights is that some really high value recommendations might be ignored by the randomness of the algorithm by chance. In practice, it's possible to mitigate this issue by hardcoding in the really desirable edges, and using seeding either the greedy or sampling algorithm with these edges. While this can work well in practice, it would be nonetheless be valuable to prove strong approximation guarantees in the weighted regime as well.

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] Robert M Bell, Yehuda Koren, and Chris Volinsky. The bellkor solution to the netflix prize.

[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 nˆ5/2 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] Purnamrita Sarkar, Deepayan Chakrabarti, and Andrew W Moore. Theoretical justification of popular link prediction heuristics. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 2722, 2011.

[25] 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.