# Efficient Fault-Tolerant Group Recommendation Using $\alpha$-$\beta$-core*

Danghao Ding, Hui Li, Zhipeng Huang, Nikos Mamoulis
The University of Hong Kong
{dhding2,hli2,zphuang,nikos}@cs.hku.hk

## ABSTRACT

Fault-tolerant group recommendation systems based on subspace clustering successfully alleviate high-dimensionality and sparsity problems. However, the cost of recommendation grows exponentially with the size of dataset. To address this issue, we model the fault-tolerant subspace clustering problem as a search problem on graphs and present an algorithm, *GraphRec*, based on the concept of $\alpha$-$\beta$-core. Moreover, we propose two variants of our approach that use indexes to improve query latency. Our experiments on different datasets demonstrate that our methods are extremely fast compared to the state-of-the-art.

## KEYWORDS

group recommendation, subspace clustering, fault tolerance

**ACM Reference format:**
Danghao Ding, Hui Li, Zhipeng Huang, Nikos Mamoulis. 2017. Efficient Fault-Tolerant Group Recommendation Using $\alpha$-$\beta$-core*. In *Proceedings of CIKM'17 , Singapore, Singapore, November 6–10, 2017,* 4 pages.
DOI: 10.1145/3132847.3133130

## 1 INTRODUCTION

Recommender systems have become standard add-ons in popular e-commerce applications and social media sites, such as Amazon, Yelp, and TripAdvisor. In a rating-based recommender system, users rate items by giving them scores. Given a *target* user $u_q$, the goal of *individual recommendation* is to estimate the *unobserved* scores of items by $u_q$ and recommend to $u_q$ the items with the highest predicted scores. Collaborative filtering (CF), the most widely used method for this problem [1], predicts unobserved scores based on the ratings of the most similar users to $u_q$. A more general problem is *group recommendation*, where users form groups and items are recommended to these groups. For example, Meetup[1] helps users to organize and participate in group activities and goodreads[2] sets up book-reading groups.

Clustering-based group recommendation [7] adopts full-dimensional clustering to organize similar users into groups. Each user is modeled by the vector of ratings given by her to all items and cosine similarity or Pearson coefficient is used as a similarity measure. However, clustering is not effective when different dimensions are relevant for different groups [8]. To address this issue, *fault-tolerant group recommendation*, which adopts subspace clustering algorithms [5] to group users, has been recently proposed [4, 8]. Subspace clustering algorithms find groups of similar users in dimensional subspaces (i.e., subsets of items [5]). Users in each projected cluster are similar considering only some dimensions, which is intuitive in real life. For example, two users may have similar taste in comedies, but a different one in Sci-Fi movies. Hence, they may belong to same cluster considering some properties of comedies, but in different clusters w.r.t. Sci-Fi properties.

Fault-tolerant group recommendation has shown its effectiveness in previous studies [4, 8]. In this paper, we aim at improving its scalability. We first show the relationship between fault-tolerant group recommendation and graph search. Then, we present our method, *GraphRec*, and its variants which are based on finding $\alpha$-$\beta$-cores in the bipartite graph that connects users to items. We experimentally show that our methods perform fault-tolerant group recommendation extremely fast, compared to the state-of-the-art.

[1] http://www.meetup.com
[2] http://www.goodreads.com

* Funded by grant no. 17205015 from Hong Kong RGC.

*CIKM'17 , Singapore, Singapore*

DOI: 10.1145/3132847.3133130

## 2 GRAPHREC

Ntoutsi et al. [8] extend the concept of fault tolerance to subspace clustering and propose a new algorithm (FTSC) for group recommendation. FTSC allows each user (item) in one subspace cluster to have $O_I$ ($O_U$) missing values, while the total missing values in one subspace cluster are controlled by another parameter $O_E$. This way, FTSC introduces more flexibility to subspace clustering based group recommendation. However, FTSC shows poor scalability due to the maintenance of missing values in each subspace cluster. We observe that the concept of fault tolerance in group recommendation can be mapped to a graph search problem through $\alpha$-$\beta$-core. This allows us to develop GraphRec, a much faster method compared to FTSC.

### 2.1 Bipartite Graph and Definition of $\alpha$-$\beta$-core

We start by presenting the concept of $k$-core [9]. Let $G = (V, E)$ be an undirected and unweighted graph, where $V$ is the set of vertices and $E$ be the set of edges. For a vertex-induced subgraph $H \subseteq G$, we denote by $\delta_H(u)$ the degree of vertex $u$ in $H$ and by $\gamma_H(u)$ the number of vertices that do not connect to $u$. We use $|V|$ to indicate the cardinality of $V$, i.e., the number of vertices in $V$.

Definition 1. *A subgraph $H = (V', E')$ induced by $V' \subseteq V, E' \subseteq E$ is called a k-core candidate, if $\forall v' \in V', \delta_H(v') \geq k$. We say a k-core candidate is a k-core if $H$ is maximal with respect to k, i.e., $\nexists H', s.t. H \subset H'$ and $H'$ is a k-core candidate.*

We observe an equivalence relationship between in-class connectivity and fault-tolerance of missing values in a fixed-sized graph $H$; for each vertex, $\delta_H(u) + \gamma_H(u) = |V| - 1$. Then, the problem of finding degree-wise fault tolerant subspace clusters becomes

similar to that of finding $k$-cores. However, it may be difficult to set $k$, since the tolerance thresholds are different between users and items. We further transfer the $k$-core paradigm to a bipartite user-item recommendation system that requires more constraints than one single parameter $k$. Instead of the matrix form in which fault tolerance subspace clustering methods operates, we work on user-item rating data represented by an undirected and unweighted bipartite graph:

Definition 2. *Let $U$ be the set of users and $I$ be the set of items. Unweighted edge set $E$ is the collection of <$U$, $I$> pairs where each user $u_p \in U$ has rated item $i_q \in I$. The bipartite graph $G$ is constructed as $G = (U, I, E)$.*

Notice that in the bipartite graph, we do not count user-user or item-item edges as missing edges for $\gamma_H(u)$. Hence, we propose the definition of $\alpha$-$\beta$-core that is applicable to bipartite graphs:

Definition 3. *A subgraph $H = (U', I', (U', I')|E)$ induced by $U' \subseteq U$ and $I' \subseteq I$ is called a $\alpha$-$\beta$-core candidate, if $\forall u \in U, \delta_H(u) \geq \alpha$ and $\forall i \in I, \delta_H(i) \geq \beta$. We say a $\alpha$-$\beta$-core candidate is a $\alpha$-$\beta$-core if $H$ is maximal with respect to $\alpha$ and $\beta$, i.e., $\nexists H', s.t. H \subset H'$ and $H'$ is a $\alpha$-$\beta$-core candidate.*

## 2.2 Mapping of Fault Tolerance Subspace Clustering to Graph Search

Through the following lemma, we observe that the definition of $\alpha$-$\beta$-core is a relaxation of fault tolerance subspace clustering:

Lemma 1. *Let $\Theta = \{\theta_1, \ldots, \theta_k\}$ be the fault tolerant subspace clustering model of bipartite graph $G = (U, I, E)$. Let $U_{min} = min\{|U_i| : \theta_i = (U_i, I_i)\}$, $I_{min} = min\{|I_j| : \theta_j = (U_j, I_j)\}$. Then each FTSC candidate must be a subset of $(I_{min} - O_I)$-$(U_{min} - O_U)$-core of the bipartite graph.*

Proof. We first prove the lemma from the view of users; the proof for items follows. For each user, the total number of rated items plus the number of unrated items equals the size of the entire set of items, i.e., for each FTSC candidate $\theta_i = (U_i, I_i)$ and each user $u \in U_i$, we have $\delta_{\theta_i}(u) + \gamma_{\theta_i}(u) = |I_i|$. Since $I_{min} \leq |I_i|$ and we require $\gamma_{\theta_i}(u) \leq O_I$ as the fault tolerance threshold, we have $\delta_{\theta_i}(u) \geq I_{min} - \gamma_{\theta_i}(u) \geq I_{min} - O_I$. □

If we require that the minimum number of users and items in a subset candidate cluster must be at least $U_{min}$ and $I_{min}$, respectively, to guarantee recommendation quality, the $\alpha$-$\beta$-core is a relaxation to the definition of an FTSC subcluster, where the pattern tolerance constraint is relaxed.

The cost of FTSC is mainly from dividing spaces into subspaces by the pattern tolerance constraint, since there is an exponential number of candidate subspaces. In FTSC recommendation systems, however, large enough subspace clusters are aggregated into a large pool of friends instead of using each subspace separately. Since each large enough subspace cluster must be a subset of our $\alpha$-$\beta$-core, we claim that the pool of friends in FTSC recommendation systems is also a subset of the corresponding $\alpha$-$\beta$-core. This way, our algorithm saves a lot when computing the subspace clusters.

## 2.3 Efficient Search of $\alpha$-$\beta$-cores

[2] introduces an $O(|E|)$ algorithm that efficiently computes $k$-cores in general undirected graphs. A nice property is that a higher-number core must be a subset of lower-number cores. In other words, a vertex that belongs to an $i$-core must also belong to a $j$-core where $j < i$. Then, we can reconstruct an $i$-core by traversing all vertices having shell numbers larger than $i$. The main idea is that if we recursively remove all vertices of degree less than $k$ and the edges incident to them, the remaining graph is $k$-core. During the process, we compute the *shell number* of each $k$-core:

Definition 4. *A vertex $v$ has shell number $i$ if $v$ belongs to an $i$-core but not to an $(i+1)$-core. $S_v$ denotes vertex $v$'s shell number.*

An $i$-core is thereby the subgraph induced by all vertices with shell number $\geq i$. In our implementation, we use a Fibonacci heap [3], which features constant amortized cost for decrease key operations, in order to perform fast updates of vertex orders. To efficiently compute $\alpha$-$\beta$-cores, we propose to use the shell numbers computed in $k$-core decomposition as an offline index, based on:

Lemma 2. *An $\alpha$-$\beta$-core in bipartite graph $(U, I, E)$ must be a subset of a $min(\alpha, \beta)$-core in the general graph $(U \cup I, E)$.*

Proof. In an $\alpha$-$\beta$-core, each user has rated at least $\beta$ items and each item is rated by at least $\alpha$ users. Since $\alpha \geq min(\alpha, \beta)$ and $\beta \geq min(\alpha, \beta)$, each vertex in the $\alpha$-$\beta$-core must have at least $min(\alpha, \beta)$ neighbors. □

We apply a similar strategy as [6] to iteratively prune disqualified vertices from the $min(\alpha, \beta)$-core. Algorithm 1 presents the details of indexing and pruning.

**Algorithm 1:** Generate $\alpha$-$\beta$-core

**Input:** Bipartite Graph $G = (U, I, E)$, shell numbers $S$, $\alpha$, $\beta$
**Output:** Vertices in the $\alpha$-$\beta$-core of G: $U', I'$

```
1  U' ← {u_i : S_{u_i} ≥ min(α, β)};
2  I' ← {I_j : S_{i_j} ≥ min(α, β)};
3  H ← (U', I', E'|U', I');
4  calculate δ_H(u) for each u ∈ U', and δ_H(i) for each i ∈ I';
5  repeat
6      foreach u ∈ U' do
7          if δ_H(u) < α then
8              foreach (u, i) ∈ E' do
9                  δ_H(i) ← δ_H(i) − 1;
10             U'.remove(u);
11     foreach i ∈ I' do
12         if δ_H(i) < β then
13             foreach (u, i) ∈ E' do
14                 δ_H(u) ← δ_H(u) − 1;
15             I'.remove(i);
16 until U' and I' do not change in one iteration;
17 return U', I';
```

## 2.4 Generation of Pattern Tolerant Clusters by $\alpha$-$\beta$-core, GraphRec and GraphRec*

Given a target user $u_q$, we first find the shell number $S_{q_u}$ of $u_q$. Since we only want information-rich friends, we require that the friends should contain at least the same number of ratings as the target user. That is, $\alpha = S_{u_q}$. We treat $\beta$ as a user input parameter that describes the degree of fault tolerance for items to be considered.

After calculating the $\alpha$-$\beta$-core, we find the connected components to which the query user $u_q$ belongs. This connected component is the friends set $F_{u_q}$ that our algorithm suggests for collaborative filtering. We name this algorithm as GraphRec.

However, online computation of connected components might be slow; in the worst case, all edges in the $\alpha$-$\beta$-core may have to be visited. Also, the target user itself may be pruned in the process of calculating the $\alpha$-$\beta$-core and there is a risk of empty result. Therefore, we propose approximate algorithm GraphRec* that directly sets friends set $F_{u_q}$ as the $\alpha$-$\beta$-core. Our experiments demonstrate that this approach has acceptable accuracy; at the same time it significantly reduces the search cost.

## 2.5 Fixed Beta Approximation: GraphRec**

We found that the online generation of $\alpha$-$\beta$-cores is the bottleneck of GraphRec, as it has $O(E)$ time complexity. Ntoutsi et al. [8] suggest a threshold method to filter out small sized clusters that do not provide enough information for recommendations. Inspired by this rationale, we propose GraphRec**, a variant of GraphRec that fixes parameter $\beta$. That is, we set $\beta$ as a constant $B = (1 - O_u) \cdot t \cdot |U|$, where $t$ is a parameter representing the fraction of total users that a subspace cluster must include. We propose to generate an offline index to dump all $\alpha$-$B$-cores, by storing a 2-D binary matrix $X$: $X(\alpha, u)$=true, *iff* $u \in \alpha$-B-core. Then, no graph traversal is conducted at each online query. The space complexity for this offline index is $O(m \cdot |U|)$, where $m$ is the max core number that a user can have in the graph. With this effective index, the online response time cost is reduced from $O(|E|)$ to $O(|U|)$ without compromising accuracy, as we show in Section 3.

## 2.6 Complexity Analysis

According to [2], the cost of the $k$-core decomposition algorithm is linear to the size of edges in a graph. The cost of finding an $\alpha$-$\beta$-core is the sum of two parts. In line 4 of Algorithm 1, the initialization of $\delta_H(v)$, i.e., the degree of vertices within $min(\alpha, \beta)$-core, has an $O(|E|)$ cost that is linear to the number of edges in the core. After initialization, the iterative removal of disqualified vertices costs $O(|E| + |V|)$, as each vertex and edge in the core will only be visited once. Since each vertex has at least one edge, this cost reduces to $O(|E|)$. Therefore, the overall cost of generating an $\alpha$-$\beta$-core is $O(|E|)$. The cost for generating the connected neighbors of a given query user is also $O(|E|)$, linear to the number of edges in the corresponding $\alpha$-$\beta$-core. Adding up, GraphRec and GraphRec* need $O(|V|)$ time for offline index generation and $O(|V| + |E|)$ time for online queries. The corresponding costs for GraphRec** are $O(|V| + |E|)$ and $O(|V|)$, respectively.

# 3 EXPERIMENTS

## 3.1 Experimental Setup

**Datasets.** We use five datasets from MovieLens[3], Jester2[4] and Yelp[5], summarized in Table 1. Note that ML_100K, ML_1M and ML_10M are collected samples from MovieLens with different sizes.

[3]http://grouplens.org/datasets/movielens
[4]http://eigentaste.berkeley.edu/dataset
[5]http://www.yelp.com/dataset_challenge

**Table 1: Statistics of the five datasets**

| | ML_100K | ML_1M | ML_10M | Jester2 | Yelp |
|---|---|---|---|---|---|
| #user | 943 | 6,040 | 69,878 | 59,132 | 1,029,432 |
| #item | 1,682 | 3,706 | 10,677 | 140 | 144,072 |
| #rating | 100,000 | 1,000,209 | 10,000,054 | 1,761,439 | 4,153,150 |

**Competitors.** We compare the performances of the following six methods. All methods were implemented in C++ using standard libraries and -O3 optimization flag. We translated the original Java source code of FTSC kindly provided by the authors of [8] to C++ code for a fair comparison. The experiments were conducted on an Intel® Core™ Cpu i7-4170MQ @2.5GHz machine featuring 16 GB of main memory and a 64-bit operating system.

- **FTSC.** Grid based fault-tolerant group recommendation.
- **FTSC_hybrid.** Hybrid fault-tolerant group recommendation.
- **FTSC_den.** Density based fault-tolerant group recommendation.
- **GraphRec.** $\alpha$-$\beta$-core based fault-tolerant group recommendation.
- **GraphRec*.** Relaxed $\alpha$-$\beta$-core based fault-tolerant group recommendation.
- **GraphRec**.** Indexed version of $\alpha$-$\beta$-core based fault-tolerant group recommendation.

**Task.** We perform a task of *rating prediction* to evaluate the performances of the methods mentioned above. Each rating dataset is randomly divided into a training set and a test set by a ratio of $4:1$. In the offline phase, each competitor only accesses to the training set to build their offline indices. In the online phase, the algorithms are requested to compute a predicted values of all ratings in the test set.

**Metrics.** We evaluate the performances with respect to MAE, RMSE and online/offline running time. Ratings in Jester2 are normalized into $1 \sim 5$ before calculating MAE and RMSE.

## 3.2 Results and Analysis

*3.2.1 Quality of User Recommendations.* Figures 1 and 2 show the results for the ML-100K dataset. We set the fault tolerance constraints as $O_I = 0.4$, $O_U = 0.3$ and $O_E = 0.4$. The parameters for FTSC are $minPts = 40$, $grid = 3$; the parameters for FTSC_hybrid are $minPts = 40$, $\epsilon = 0.1$; the parameters for FTSC_den are $minPts = 40$, $\epsilon = 0.1$. We set $\beta = 100$ for GraphRec and GraphRec*, while for GraphRec**, $t = 0.2$. All algorithms except FTSC_den present a similar accuracy. Regarding offline runtime, we observe that FTSC class algorithms are at a magnitude of 100 seconds, while GraphRec, GraphRec* and GraphRec** are at a magnitute of a few seconds. Regarding online runtime, we can see that our relaxation in GraphRec* improves online latency and our index in GraphRec** renders the response time competent to FTSC algorithms. These two algorithms improve the efficiency of GraphRec, while preserving good accuracy.

However, for other datasets, the offline runtimes of FTSC class algorithms exceed one day, which means that they are not feasible for large scale applications. Therefore, we do not report their results. For GraphRec and GraphRec*, we use $\beta = 100$ in all datasets and we set $t = 0.01$ for GraphRec** in all datasets. Tables 2 and

**Table 2: Offline Runtime (in sec)**

| | ML_100K | ML_1M | ML_10M | Jester2 | Yelp |
|---|---|---|---|---|---|
| FTSC | 712.503 | >1day | >1day | >1day | >1day |
| FTSC_hybrid | 610.728 | >1day | >1day | >1day | >1day |
| FTSC_den | 370.040 | >1day | >1day | >1day | >1day |
| GraphRec | 0.050 | 0.223 | 0.769 | 0.191 | 6.441 |
| GraphRec* | 0.050 | 0.223 | 0.769 | 0.191 | 6.441 |
| GraphRec** | 1.965 | 21.689 | 188.297 | 24.576 | 17.767 |

**Table 3: Online Runtime (in msec)**

| | ML_100K | ML_1M | ML_10M | Jester2 | Yelp |
|---|---|---|---|---|---|
| FTSC | 0.868 | — | — | — | — |
| FTSC_hybrid | 0.649 | — | — | — | — |
| FTSC_den | 0.017 | — | — | — | — |
| GraphRec | 5.932 | 66.221 | 461.610 | 114.690 | 89.991 |
| GraphRec* | 1.540 | 10.858 | 14.117 | 27.361 | 70.226 |
| GraphRec** | 0.381 | 4.312 | 3.022 | 12.913 | 2.877 |

**Table 4: MAE (normalized)**

| | ML_100K | ML_1M | ML_10M | Jester2 | Yelp |
|---|---|---|---|---|---|
| FTSC | 0.799 | — | — | — | — |
| FTSC_hybrid | 0.797 | — | — | — | — |
| FTSC_den | 0.946 | — | — | — | — |
| GraphRec | 0.802 | 0.731 | 0.677 | 0.836 | 0.977 |
| GraphRec* | 0.832 | 0.802 | 0.891 | 0.925 | 1.068 |
| GraphRec** | 0.834 | 0.770 | 0.815 | 0.925 | 1.032 |

3 present the runtime results. Since MAE and RMSE are equivalent norms, we only present MAE in Table 4 due to limited space. GraphRec is the most accurate algorithm in all cases, however, the query latency grows fast when size of dataset increases. GraphRec* improves query latency but the degree of improvement depends on the structure of datasets; it performs well in dense and small $|I|$ sets like Movielens datasets and Jester2, but fails to respond fast in a sparse and diversified sets like Yelp. GraphRec**, however, scales well regardless the structure of the data and remains relatively accurate compared to GraphRec.

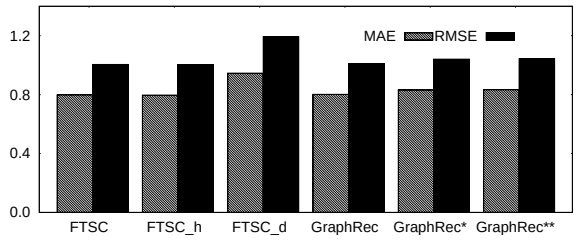
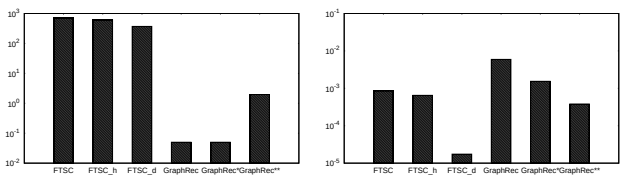



**Figure 1: MAE & RMSE for User Recommendation in ML-100k Dataset**




**(a) Offline Runtime (sec)** **(b) Online Runtime (sec)**

**Figure 2: Offline & Online Runtime for ML-100k Dataset**

*3.2.2 Quality of Group Recommendations.* Since there is no ground truth in group recommendations, we use the individual group members ratings to asses the quality. We randomly generate 1,000 user groups from the ML-100K dataset and use the average rating for commonly rated items as ground truth. Each group consists of 5 members and we test all common items for each group. The parameter settings are the same as our experiments of user recommendations. The runtimes of algorithms are linear to the runtime in user recommendations with respect to the group size. Therefore we do not report the runtime of group recommendation experiments. In terms of effectiveness, as shown in Figure 3, our methods are similar to FTSC algorithms.

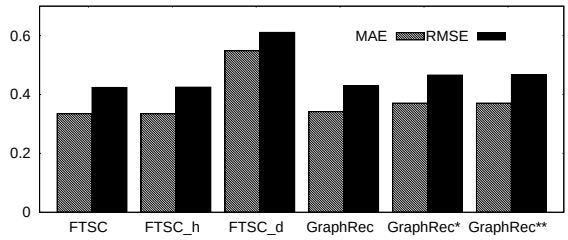



**Figure 3: MAE & RMSE for Group Recommendation in ML-100k Dataset**

## 4 CONCLUSION

In this paper, we show the connection between fault-tolerant group recommendation and graph search. Based on it, we propose an efficient fault-tolerant group recommendation method, GraphRec, and its two variants which are based on the concept of $\alpha$-$\beta$-core. Our experiments demonstrate the efficiency of our solutions, compared to the state-of-the-art. In the future, we plan to investigate the adoption of other graph structures to approximate $\alpha$-$\beta$-cores, in order to further improve efficiency. Another interesting direction is to scale GraphRec algorithms up by applying a distributed computing setting.

## 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. Knowl. Data Eng.*, 17(6):734–749, 2005.
[2] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.
[3] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *FOCS*, pages 338–346. IEEE Computer Society, 1984.
[4] S. Günnemann, E. Müller, S. Raubach, and T. Seidl. Flexible fault tolerant subspace clustering for data with missing values. In *ICDM*, pages 231–240. IEEE Computer Society, 2011.
[5] H. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *TKDD*, 3(1):1:1–1:58, 2009.
[6] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. *Computer networks*, 31(11):1481–1493, 1999.
[7] E. Ntoutsi, K. Stefanidis, K. Nørvåg, and H. Kriegel. Fast group recommendations by applying user clustering. In *ER*, volume 7532, pages 126–140, 2012.
[8] E. Ntoutsi, K. Stefanidis, K. Rausch, and H. Kriegel. "strength lies in differences": Diversifying friends for recommendations through subspace clustering. In *CIKM*, pages 729–738. ACM, 2014.
[9] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.