This project is an attempt to generate a meaningful network from Amazon product co-purchases and product reviews; understand the community structure and the kind of clusters generated. And if they imply anything useful. Also, this was an attempt to identify if any simple network metric could help us understand the relationship between various products.

All of the sample code and derived data-sets are present at <https://github.com/MoonieMama/SNA-project>

# Obtaining data

## Co-purchase network

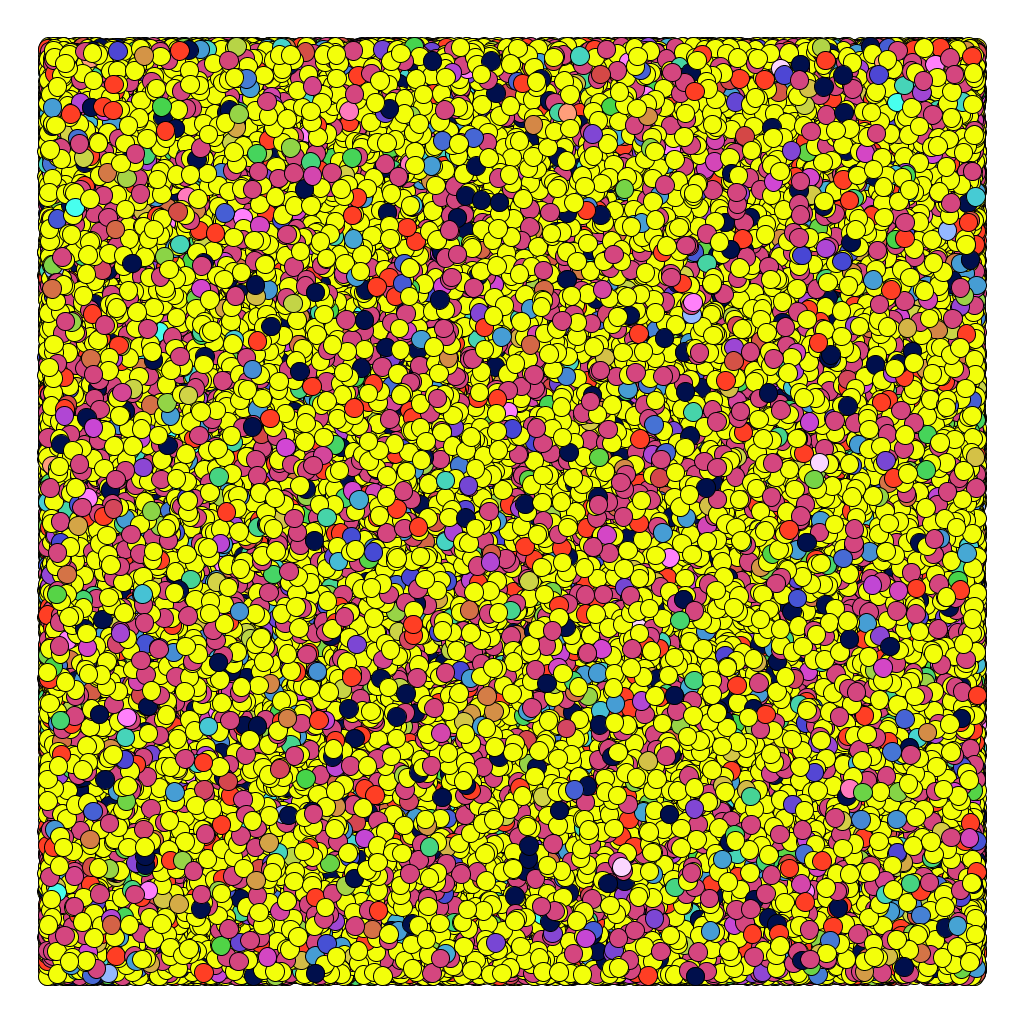
Amazon co-purchases reflect whether customers who buy a given product frequently purchase another – if so, the dataset we have contains a directed edge between the two products. The dataset can be found at <http://snap.stanford.edu/data/amazon0302.html> . Additional metadata has been stripped, and the file has been used as the corresponding edgelist for analysis. Please refer to the **readGraph - Copy.r** file for the exact details.

The network has 262,111 products and 1,234,877 co-purchase edges. As such, it was extremely difficult to load the file into Gephi (and even Graphviz) and perform any visualization – I had to utilize igraph and network X for any computations.

## Review network

We also have a set of product metadata, cataloguing product information, categories and customer ratings for a particular product. The dataset is derived from <http://snap.stanford.edu/data/amazon-meta.html> . As pre-processing, I have restricted myself only to the products ids found as part of the co-purchase network.

As part of parsing the file, I’d initially utilized the **parse** method in **parsemeta.py** to extract the id, title and category attributes (to a **pid**.txt and a gdf file). The gdf file (w/o loading edges) was coloured based on product category (Book, DVD, music, video):



(Books: 62%, Music: 22%, DVD: 6%, Videos: 10%)

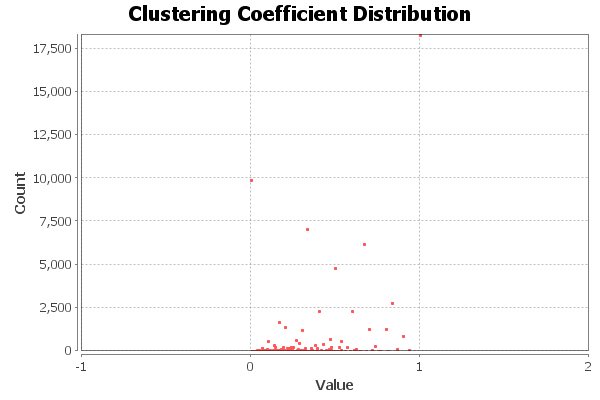
The second parse method (**parseCustomer**) was utilized to create a weighted bipartite graph having edges from customers/reviewers to the product, with edge weights reflecting the rating provided to the product. Unfortunately, the generated edgelist (refer to **pid\_cid\_edges\_weighted2** in the repository) couldn’t be loaded up into igraph (thanks to alphanumeric vertex id’s), so further analysis of this graph was carried out using networkX. As such, this graph had 1218028 vertices and 2988683 edges.

# Analyzing co-purchase network

N = 262,111

E = 1,234,877

Degree distribution: Refer to the report in the **degDist** folder.

Transitivity: 

The global transitivity value (ratio of triangles and connected triples) = 0.23

The average clustering coefficient (over all vertices) = 0.42

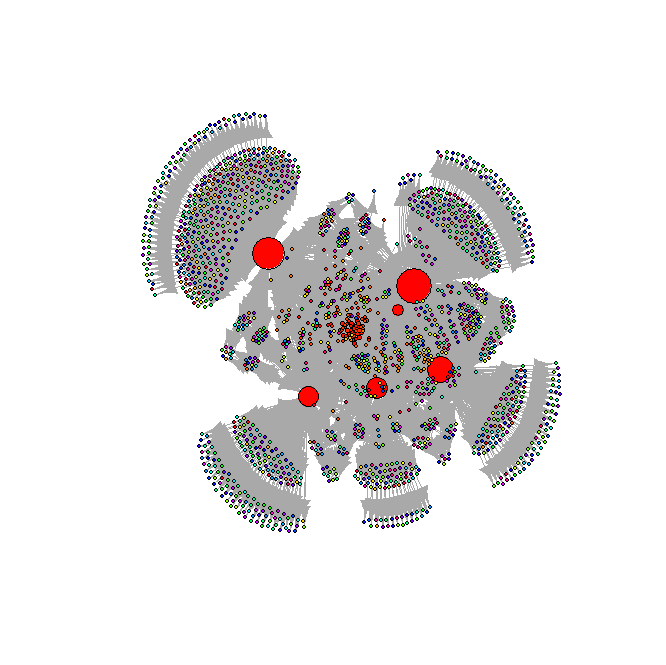
This is much higher than the same metric over an ER graph with the same number of nodes (refer to **Clustering(Random)** folder)

Given that, I tried to run 4 community finding algorithms over this particular graph: fast-greedy (undirected), infoMap (directed), label propagation (undirected) and walktrap (undirected). Also, following the creation of communities, I had compressed nodes belong to one community into a giant node and tried to construct the existing network after removing self-loops. The resulting graphs were laid out using either the lgl, fruchterman.reingold.grid or rarely, fruchternam.reingold (due to the extremely large number of nodes, and the fact that standard spring force layout algorithms take *cubic* time in the number of nodes). Refer to <http://www.inside-r.org/packages/cran/igraph/docs/layout> for a better understanding of the above. The sample code for achieving this is part of the **readGraph – Copy.r** file.

### Fast greedy:

The fast greedy algorithm is a hierarchical, bottom-up approach which tries to modularity, by greedily combing vertices having different labels into a single union until there can be no further increase in modularity.

Results are documented in the **FastGreedy** folder. Also included is a mention of the highest betweenness and closeness centrality node in the compressed cluster graph.



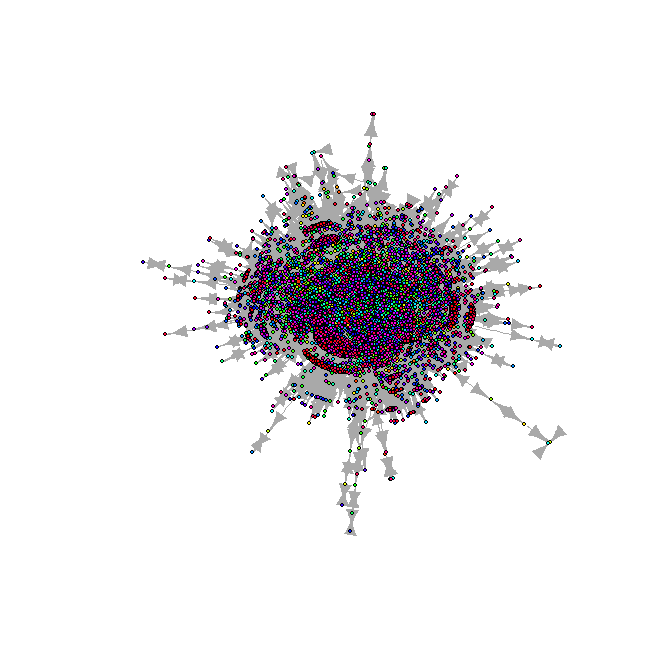
The findings of the community include highest modularity among all clustering algorithms, large cluster sizes among the top 4 clusters, and several topics/product types among these. It was rather difficult to assign a product description to the large clusters – however, there are several subtopics which can be seen prominently lie in these, and this occurs frequently as the community size goes down. For instance, the 13th community can be noted to have a community on music products occurring along with books of various sub-genres (autism, mystery). The smaller clusters too have a wider smattering of products, but it isn’t rather clear as to how the split occurred. The intra cluster density was seen to be more concentrated in the largest clusters, and the smaller clusters had a small spread much below the value in the larger clusters.

Another interesting aspect I barely touched upon was utilizing different community detection algorithms on the subgraph induced by the clusters generated. For instance, on community 11 generated from fast-greedy, I ran the label propagation community and the walktrap community finding algorithms – leading to a result set which was somewhat closer to having cliques in terms of product types (and increasing modularity). Refer to **FG11LPCommunities**, **FG11WCCommunities**, and the detailed files under **FastGreedy** folder.

### Walktrap:

This algorithm is based on the basis of random walks of short steps. The idea is that a random walk of short length would tend to stay inside a particular community.

The results are documented in the **Walktrap** folder.



As noticed, the modularity is only lower to fast-greedy, but there are several small-sized clusters in this network. The product differentiation is hard to make, but it seems to have a better separation as compared to fast-greedy. Refer to the file for community 16 for understanding how music products are clustered in a better fashion (for religious and classics). The intra cluster density is spread more uniformly than fast greedy, but there is still a large deviation in the values for the largest clusters.

### Label propagation:

This is a simple approach in which every node is assigned one of k labels. The method then proceeds iteratively and re-assigns labels to nodes in a way that each node takes the most frequent label of its neighbors in a synchronous manner. The method stops when the label of each node is one of the most frequent labels in its neighborhood – and takes near linear time for completion

Refer to the **LabelPropagation** folder for results.

We have the lowest modularity score assigned to this split – and the intra cluster density is much smaller and more uniform. Here, the labelling in smaller communities is much better than in the previous cases – but it is still hard to assign a label to a cluster super-node.

### InfoMap(directed):

Info Map is a clustering algorithm based on an information theoretic approach which builds a grouping which provides the shortest description length for a random walk on the graph.

Results are in the **InfoMapDirected** folder.

This approach results in a relatively high modularity for the clusters – and small cluster sizes. The clusters seem to have a better classification of product type, and product category (introductory and guide resources), but they still defy proper classification. Yet, the ordering of categories is uniform with smatterings of random topics. The intra cluster density seems to be uniformly spread – much more than the other clustering.

## Comparing communities:

I used the compare communities’ method igraph to see a scalar measure of how different the network communities formed through the different algorithms are. The scalar measure utilized is an information-theoretic approach which is called the variation of information metric introduced by Meila (2003): this variation essentially measures the amount of information lost and gained in transforming from one cluster to another. The measure can be a real number bounded by the logarithm of the number of nodes in the clusters, and depends on the conditional entropy of the two clusters, based on a simple discrete random variable for clusters. The smaller the number, the closer are the clusters – in terms of the information encoded. For an in-depth treatment, refer to the paper <http://www.stat.washington.edu/mmp/Papers/compare-colt.pdf>

The results are stored in the CompareClustering text file.

What we can see is that infoMap and label propagation are relatively similar, but the communities generated by fast greedy and walktrap represent a great deal of change as compared to the former two.

# Analyzing review network

N = 1,218,028

E = 2,988,683

The review network was formed by creating directed edges from reviewers to the product, based out of the zipped Amazon meta text file data set. This was created through the **parseCustomer** function in **parsemeta.py**, and stops at creating additional edges beyond the number of customers in in the co-purchasing network. The idea was to see if a simple network metric could be representative of similar products, and maybe of co-purchasing. Note that we have a caveat here: namely, we’re potentially utilizing reviewer ratings as representative of the frequency of purchases for a pair of products – but intuitively, it was easier to link these two since a high rating by an influential reviewers on two products could lead to them being purchased together.

Given a reviewer – product digraph, I’d utilized a simple metric of highest weighted in-degree for products for reviewers who reviewed a product, in order to find similar suggestions for co-purchases. Again, there may be issues just considering reviewers as representative of the products they buy – but it seems that reviewers too can be thought of as communities, based on the type of products they buy.

I didn’t do an in-depth community creation and tested out products recommendations for individual products, the data for which is present as part of the **2reco\_1** and **1reco\_1** txt files, for products with ids 2 and 1 respectively. This was coded up in the **Reviews.py**, and the operation performed was as follows:

For a product (with id 2, say “Candlemas: Feast of Flames,Book” – categorized on Goodreads to be in the pagan, wiccan, witchcraft and occult categories), the list of all reviewers of this product were extracted, and all of the products these reviewers had reviewed. Once we had this induced subgraph, we ordered the products based on a simple heuristic of a linear combination of the weighted in-degree in the parent graph and the weight in-degree in the sub-graph and created a recommendation list text file, sorted for all products.

Refer to **2reco\_1** and **1reco\_1** text files to get data for recommendations, and match them with the corresponding id in the **pid.txt** file to get the product name.

The interesting part about this was that the first few products (for instance, “Harry Potter and the Sorcerer’s Stone” and “The Exorcist”) were popular products which may (or may not have had to do with the product in hand – but they did have elements of similarity (occult and magic). As we went down the list, we find products having categories that match closely to the considered product (“American Gods” – old and new gods), (“To Ride a Silver Broomstick: New Generation Witchcraft” – witchcraft), (“The Little Prince” – fantasy), (“Lord of the Rings”), and (“Celtic Wicca: Ancient Wisdom for the 21st Century”). It may be easy to make further adjustments for accounting for the best similar products, by searching in the range of the reviewers’ ratings in the product list, and extend it onwards. We have some reason to believe that we have utilized some notion of the users tending to have similar interests (assuming reviewers reviewed products they’d bought, making them accountable as a co-purchase).

A potential drawback is that I haven’t utilized the helpfulness of the user rating (which are classified as total and helpful votes in the data set). This shouldn’t be too hard to utilize. Another interesting thing to do is a 1-d projection of this graph, by connecting products which have similar reviewers, and weighing the edges with some measure of mutual information about the reviewers ratings for this pair of products and all products the reviewers had rated. It would be nice to analyze this network with the actual co-purchase network, and see measures of clustering in the latter.