Social graph clustering

Igor Kabiljo

***Contents* — In this paper, we will work on a problem of social graph clustering. Problem of graph clustering is very well studied, but in almost all cases, disjunctive and total clustering is created. In a social graph, it is obvious that we can have people that belong to many groups, as is the case for the majority of people. But we can also have a person that does not belong to any group. We devise few algorithms that can be used to solve this problem. Also, currently, there is no good general metric for measuring quality of such clustering, so we created one that best suits the needs of specified task.**

***Keywords* — clustering, eigenvectors, minimal cost spanning tree, quality, spectral graph theory, social graph**

I. Introduction

SOCIAL networks, such as Facebook, Twitter, LinkedIn, etc, are becoming very popular in the last few years. There are quite a few networks with more than one million users, with Facebook currently leading with more than 500 million users. There is a lot of hand created data in those networks, and there are a lot of ways one can use that data to extract interesting information. For some types of information, a point that they are hand created makes them extremely valuable.

In most social networks, each user defines a list of other users that he is connected to. These connections make a graph that we will call social graph. On Facebook, each user defines a list of people that are their friends, with a constraint that such connection has to be confirmed. Requiring confirmation makes Facebook's friendships graph very accurate.

Each person has friendships from many spheres of their life. For example typical user can have friends from high school, university, work... The question is, can we automatically detect these groups, from the data user had

Igor Kabiljo, Računarski Fakultet, Srbija (email: ikabiljo@gmail.com, phone: +381-11-2697-991)

already provided. Specifically, in this paper, for any given user we will use only the subgraph of the social graph induced by his friends, and try to extract all of these groups from it. In purely graph theoretic terminology, we will try to create clusters of a given graph that will best possibly satisfy some conditions that arise naturally.

Most of the available clustering algorithms create a full disjunctive clustering, which means each vertex will be in exactly one cluster. In reality, friends can be in multiple groups or in no group at all. So we need a clustering algorithm where some vertices can be in more than one cluster, and some in none. There is very little work on overlapping clustering so far. We will describe few approaches to this problem, few algorithms for solving it, make a comparison both on automatically generated as well as actual social graphs, and give directions for future work.

II. TASK

Given a set of vertices V, and a set of undirected edges E between them, we want to find a set of clusters $S = \{C_1, C_2 \dots C_k\}$, where each C_i is a subset of V, that will maximize quality function $quality(S)$.

Without explicitly defining quality function, above statement does not mean much. The quality function is an essential part of a given task statement, since different quality functions will give different resulting clusters, and so could require different algorithms. There is not a generally accepted quality function for clustering; so many clustering papers give different functions that suit their specific task. None of these seem suitable enough for our particular task, so we will have to define our own quality function. We will start by describing quality function as intuitive quality of a clustering, and try to represent that intuitiveness as well as possible. Clustering that is intuitively good creates groups that represent actual groups of people in real life. Any clustering should satisfy that each cluster is internally dense, and externally sparse, and that clusters are not too similar. We are also not interested in very small clusters, but that is a somewhat arbitrary requirement.

III. QUALITY FUNCTION

Let us try to derive a good quality function. First we will see what makes a single cluster a good one, and then we will extend it to cover a set of clusters. So we have a set of vertices $V = \{v_1, v_2 \dots v_n\}$, a set of edges $E \subset V \times V$, and a set of clusters $S = \{C_1, C_2, \dots, C_k\}$.

The first goal is to make cluster internally dense. One good measure of internal density of a cluster is the ratio of edges present in induced subgraph,

compared to a clique of the same size. We will introduce α, the internal density coefficient, that will be calculated by:

$$\alpha(C) = \frac{2|E(G[C])|}{|C|(|C|-1)}$$

where $E(G[C])$ is standard notation that means induced subgraph.

Next, we want to make a cluster externally sparse. First let us denote maximal number of inbound edges with:

$$b(C) = \max_{v \in V \setminus C} |\{u | u \in C, (u,v) \in E\}|$$

Now we can denote external sparsity with β, and calculate it as the maximal ratio of inbound edges for any outer vertex, or:

$$\beta(C) = \frac{b(C)}{|C|} = \max_{v \in V \setminus C} \left(\frac{|\{u | u \in C, (u,v) \in E\}|}{|C|} \right)$$

Note that both α and β are always in interval $[0,1]$.

Next we need to put together formula for a set of clusters. We want to keep as many of good clusters as possible, but we do not want almost identical clusters, that differ only on a few vertices. Also we do not want some cluster to be a subset of another cluster, though this requirement is not necessarily needed. We define the similarity index as:

$$similarityIndex(C_i) = \sum_{j \neq i} \left(\frac{|C_i \cap C_j|}{|C_i|} \right)^2$$

The lower the $similarityIndex$ is, the better it is, since a cluster is more different from the other clusters.

Also, we don't want α and β to be too close and we don't want clusters to be small, so we add artificial penalty based on the size of a cluster:

$$sizePenalty(C) = \frac{2}{|C|},$$

With this, the only possible cluster of size less than 4 is clique of size 3 that is a connected component of G, in other words has no edge to the rest of the graph.

Now let us put all this together:

$$quality(S) = \sum_{C \in S} \alpha(C) * \big(1 - similarityIndex(C)\big) - \beta(C) - sizePenalty(C)$$

This formula assures that it is best to take as many good and different clusters as possible.

IV. CLUSTERING ALGORITHMS

We will present here two approaches to solving our problem. Both are recursive, with different recursive step, but same recursion pattern.

Recursive step in clustering should, for any given graph, say that either the graph as a whole makes one cluster, or it can be partitioned in a set of subparts, such that any cluster is contained in one of the subparts. So if the graph has the best clustering $S = \{C_1, C_2, \dots, C_k\}$, then partition P= $\{Q_1, Q_2, \dots, Q_l\}$ should satisfy $(\forall C_i \in S)(\exists Q_i \in P)(C_i \subset Q_i)$ and $V \neq Q_i$.

Any recursive step can trivially be extended to (recursive) graph clustering algorithm, by simply recursively applying recursive step to all subparts returned by recursive step itself. That means that the problem becomes: finding a partition of a graph that will preserve all clusters of the initial graph, if such partition exists. This simplifies the problem, and both algorithms we will present here will define a recursive step, and then extend it to recursive graph clustering algorithm.

A. MCST Splitter

Let us assume we have a function $f(u, v)$, that in some sense represents how much do nodes u and v need to be in the same cluster. The larger value means nodes u and v should be in the same cluster; and the smaller that value is means they should not be in the same cluster. Assuming that we have such a function, we will now create an algorithm for the recursive step, that partitions a given graph. First, we create complete weighted graph G', with edge (u, v) having weight $f(u, v)$. Now, we want to find partition P, such that all pairs of vertices u and v that are not together in the same part of the partition, have minimal $f(u, v)$. One way to define it strictly, is to say that maximal $f(u, v)$ for all such pairs u v is minimal possible, from all partitions P. We can now say that if in such partition, maximal $f(u, v)$ is larger than some threshold, then V is a cluster, otherwise we can partition it into P.

First, let us try to create a best partition P, in which no vertex is in more than one partition. Obviously, best partition will have exactly two parts, $P = \{Q, R\}$, because if we had more than two, joining two of them will not increase maximal $f(u, v)$. So maximal $f(u, v)$ becomes $\max_{u \subset Q, v \subset R} f(u, v)$. We can calculate that easily, by creating maximal cost spanning tree in graph G', and then removing minimal edge from that tree. Then we get two trees, that are not connected, and we take them as partition into Q and R.

If we want general overlapping partitions, the problem becomes much harder. We can create an approximation, by removing 2 minimal edges from maximal cost spanning tree, thus getting three subtrees T_1, T_2 and T_3. Without loss of generality, let us say that two removed edges were connecting T_1 and T_2, and T_2 and T_3 respectively, and the edge between T_1 and T_2 had smaller

weight. Now we can create a partition $P = \{Q, R\}$, such that $Q = T_1 \cup T_2, R = T_2 \cup T_3$. Now, maximal $f(u, v)$ becomes $\max_{u \in T_1, v \in T_3} f(u, v)$, and we know that it must be at least as good as disjunctive partition since:

$$\max_{u \subset T_1, v \subset T_2 \cup T_3} f(u, v) = \max\left(\max_{u \subset T_1, v \subset T_2} f(u, v), \max_{u \subset T_1, v \subset T_3} f(u, v)\right)$$

Now let us see what we can use for function f. As we said, it is necessary that the larger the value gets, the chance that vertices should be in the same cluster increases. The easiest function that obviously satisfies that criteria is $f(u, v) = 1$ if $(u, v) \in E$, otherwise 0. It is obvious that if two vertices are connected, the chance of them being in the same cluster is larger than if they are not connected. More precise function can be:

$$f(u, v) = |(x|(u, x) \in E \wedge (x, v) \in E)|$$

or in other words $f(u, v)$ is the number of mutual neighbors. Notice that the first definition represents the number of paths of length one between u and v, and the second definition represents the number of paths of length exactly two between u and v. We can obviously expand this, and for any positive k, create a function that represents number of paths of length exactly k between two vertices.

B. Spectral clustering

Currently, most popular clustering algorithms are based on spectral graph theory. They mostly work in combination with *k-means* algorithm, and give disjunctive partitioning. We will try to create overlapping partitions using spectral graph theory.

For a given matrix A, vector v that satisfies $Av = \lambda v$ is called eigenvector, and λ is called corresponding eigenvalue. Eigenvector represents a vector that preserves its direction when multiplied by matrix A. All values λ for which eigenvector v exists (so all eigenvalues) represent all solutions to the equation $\det(A - \lambda I) = 0$. For a symmetric matrix A, all eigenvectors are mutually orthogonal, there are exactly n of them, and all eigenvalues are real. For our purposes, we can sort all pairs (v, λ) by eigenvalue in descending order, and we will enumerate them by this ordering. So when we say first eigenvector, that will mean eigenvector that corresponds to the largest eigenvalue. If there are multiple eigenvalues that are equal, we can sort them in any order.

Since we are starting with a graph, we need to construct a matrix that will represent it. There are multiple ways of creating it, so that its eigenvectors suit clustering best. We will use two of these. Based on a graph, we can create an adjacency matrix A, where:

$$a_{i,j} = \begin{cases} 1, & (v_i, v_j) \in E \\ 0, & (v_i, v_j) \notin E \end{cases}$$

And we can also create *Laplacian* matrix L, defined by:

$$l_{i,j} = \begin{cases} -a_{i,j}, & i \neq j \\ \sum_k a_{i,k}, & i = j \end{cases}$$

For both matrices, we will calculate their eigenvalues and their respective eigenvectors. For adjacency matrix, for clustering, eigenvectors that correspond to largest eigenvalues, skipping first, are interesting (those at positions $2, 3, ...$). For Laplacian, eigenvectors that correspond to smallest eigenvalues are used, again skipping first (those at positions $n-1, n-2, ...$). These eigenvectors are known to give good results when used in graph clustering. For Laplacian there are theoretical proofs of some properties that make clustering work so good. For adjacency matrix, on the other hand, there are no such proofs, but it empirically gives good results. The eigenvector that we are skipping is not interesting for this purpose, but can be interesting for others. For adjacency matrix, the largest eigenvalue is always equal to 1, and for Laplacian, the smallest eigenvalue is always equal to 0. Good overview of current results in graph clustering based on spectral theory can be found in [1]. Good broader overview of spectral graph theory can be found in [2].

Here, we will only use the first of these interesting eigenvectors (so $2nd$ for adjacency matrix, and $n-1$ for Laplacian, for which corresponding eigenvalue is called algebraic connectivity), and try to cluster based on it. We will sort all vertices by its corresponding value in that eigenvector, and use that ordering. Let's enumerate vertices after ordering $w_1, w_2, \dots, w_n$.

Known approach for creating disjunctive partition of a given graph is to try to split all the vertices in two consecutive (in calculated ordering) lists of vertices. In other words, for a splitting at index s, we will divide vertices in two parts $Q = \{w_1, w_2, \dots, w_s\}$ and $R = \{w_{s+1}, w_{s+2}, \dots, w_n\}$. We can measure the quality of partition in two disjunctive parts as a ratio of edges present that go from Q to R, in other words $|\{(u,v)|u \in Q, v \in R\}|/|Q|/|R|$. We can now simply calculate that ratio for all indexes s, and partition where the ratio is the lowest. If that lowest ratio is above some threshold, then we can say that current graph is a cluster, and does not need more partitioning.

But we want to split in not necessarily disjunctive partition. One way is to find two indices $s_1 < s_2$ that have lowest ratio. If we denote with $T_1 = \{w_1, w_2 \dots w_{s_1}\}$, $T_2 = \{w_{s_1+1}, w_{s_1+2} \dots w_{s_2}\}$, and $T_3 = \{w_{s_2+1}, w_{s_2+2} \dots w_n\}$, our partition will be $P = \{T_1 \cup T_2, T_2 \cup T_3\}$. We can also add requirement that distance between s_1 and s_2 must be in some allowed interval, so that we don't split into two almost distinct partitions, and that we do not split in two

partitions that have almost all vertices in common. If the larger of these two ratios is higher than threshold, then we will still split in only disjunctive partitions.

Let us now see if we can create a better overlapping partitioning using spectral ordering. As noted in previous section, given method for calculating quality of partitions has serious deficiencies, and works properly only for disjunctive partitions. For overlapping partitioning our restrictions, on what possibilities are tested, improve quality of partitions. We need a better measure of quality of partitions. One aspect of ordering by eigenvector, that we have not mentioned so far, but will help us here, is that it separates one cluster at the time. This means that we could expect to have one of two partitions we want to create in the resulting cluster itself. And we know a good quality function for clusters. For one part we want it to be good cluster C, but what do we require for the rest R? If we just divide our graph into two disjunctive parts, C and $R = V \backslash C$, we would miss clusters that overlap both C and $V \backslash C$. So, we want vertices that have many edges to be added to R. In other words, we want R to have external sparsity as low as possible. Internal density is a plus, but not that important. Also we do not want too much overlap between C and R, so that the recursion does not go too wide, and do a lot of repeated work. In total, we are looking for something just like $quality$ function, so we are going to use it. The problem then becomes, finding partition that has largest possible $quality$. We will again trust in sorting by eigenvectors, and only check consecutive parts, partitioned by two indexes s_1 and s_2, and having same partitions as in previous overlapping algorithm. For each pair of indexes, we will check its $quality$ and take one with largest value.

C. Improving

Now that we have a way to create clusters of a large graph, we should also look if those solutions are locally optimal, or if there is an easy way to improve them, by adding or removing few nodes, for example. This is especially important, since in our algorithms there are not a lot of places where quality function is directly used. Both given algorithms and defined quality function use "logical" meaning of good clustering, so they should not be too far off. But constraints like non-similarity of clusters intentionally are not in any of the algorithms, so that they find all possible different clusters, and then in improving step we can choose which combination gives best value.

We can first locally improve all clusters separately, and then pick best combination out of those.

If we want to calculate quality of a single cluster, the formula becomes

$quality(C) = \alpha(C) - \beta(C) - sizePenalty(C)$. We can iteratively improve this value by adding or removing one element to the current cluster in each step. We can try all possibilities, so checking which cluster, out of $\{C \cup v | v \notin C\} \cup \{C \backslash v | v \in C\}$ possible close clusters, has largest $quality$, and do that iteratively. When all possible clusters have $quality$ smaller then current cluster, we have found local maximum. Since we are always strictly increasing current $quality$, we know that this process must end. We can also add a possibility of swapping two vertices, one to be removed from the cluster, and other one to be added.

When we have set of clusters, that we don't want to change internally, we can try to choose best possible combination of those clusters, that will have the largest possible $quality$.

Since we don't expect large number of clusters, and most clusters should be good, this shouldn't be a hard problem, so we can approach it greedily. We can add, one by one, best cluster that hasn't already been selected, best in a sense that when added to current set, it will give largest possible $quality$. We add clusters until any new cluster will decrease the current value.

V. Application

Wowd is a social media tool that helps you organize your online social life and discover the best of Facebook and the entire Web. One of the most used features is social integration with Facebook, which allows users to stay organized and cut through the clutter of information Facebook has. And since clutter is getting worse each day, that is becoming more important. One of the Facebook features targeted at both reducing information overload, and resolving privacy issues, are Groups. They allow users to create separate groups, and then have all the conversations in it separated from other conversations. In that way, you both have ability to share information only with group of people that information is targeted for, and you can stay organized. One major problem with this feature is that users need to create those groups. Facebook was hoping that at least one person in each group will be willing to spend time and create a group, but it looks like Groups are not used much. Wowd has solved this problem by automatically creating those groups for you, calling them SmartFeeds. Anybody can go to wowd.com and check how is, best algorithm given here, performing on his own social graph.

VI. Conclusion

We have worked on a problem for which there is some previous work, but not much. General graph clustering has been well studied, but those algorithms don't work well on our task. We have given few approaches to solving our problem, and given a novel way for comparing the qualities of different clusterings. Our way for calculating qualities of clusterings can be easily used in comparing results of any set of algorithms. Results show that best approach uses spectral clustering with best overlapping partition based on same quality formula, and it gives better results than previous work in this area.

VII. Works Cited

1 Luxburg UV. A tutorial on spectral clustering. Statistics and Computing. 2007;17(4).

2 Cvetković D, Rowlinson P, Simić S. An Introductoin to the Theory of Graph Spectra. 2010.

3 Meglicki Z. The Householder-QR/QL Algorithm. [Internet]. Available from: http://beige.ucs.indiana.edu/B673/node30.html.

4 Baumes J, Goldberg M, Krishnamoorthy M, Magdon-Ismail M, Preston N. Finding Communities by Clustering a Graph into Overlapping Subgraphs. 2005.

5 Mishra N, Schreiber R, Stanton I, Tarjan R. Clustering Social Networks. 2007.

Abstract

U ovom radu ćemo se baviti problem klasterovanja socijalnog grafa. Ovaj problem je veoma dobro istražen, ali u skoro svim slučajevima radi se o potpunoj podeli na disjunktne klastere. Očigledno je da u socijalnom grafu ljudi najčešće pripadaju u više grupa. Ali takođe imamo i ljude koji ne pripadaju nijednoj grupi. Opisaćemo nekoliko pristupa ovom problemu, i na koji način se problem može rešiti. Takođe, ne postoji nijedna dobra mera koliko je određeno klasterovanje dobro, na osnovu koje bi se mogli upoređivati algoritmi, tako da smo definisali meru koja najbolje odgovara.

Klasterovanje socijalnog grafa

Igor Kabiljo