Ego-net Community Mining Applied to Friend Suggestion

Alessandro Epasto[*]
Brown University
epasto@cs.brown.edu

Silvio Lattanzi
Google, New York
silviol@google.com

Vahab Mirrokni
Google, New York
mirrokni@google.com

Ismail Oner Sebe
Google, Mountain View
sebe@google.com

Ahmed Taei
Google, Mountain View
ataei@google.com

Sunita Verma
Google, Mountain View
sunitav@google.com

ABSTRACT

In this paper, we present a study of the community structure of ego-networks—the graphs representing the connections among the neighbors of a node—for several online social networks. Toward this goal, we design a new technique to efficiently build and cluster all the ego-networks of a graph in parallel (note that even just building the ego-nets efficiently is challenging on large networks). Our experimental findings are quite compelling: at a microscopic level it is easy to detect high quality communities.

Leveraging on this fact we, then, develop new features for friend suggestion based on co-occurrences of two nodes in different ego-nets' communities. Our new features can be computed efficiently on very large scale graphs by just analyzing the neighborhood of each node. Furthermore, we prove formally on a stylized model, and by experimental analysis that this new similarity measure outperforms the classic local features employed for friend suggestions.

1. INTRODUCTION

The advent of online social networks has opened new opportunities to analyze the evolution and the structural properties of social networks at macro and microscopic level. Researchers have now access for the first time in human history to large online social network datasets that can be easily analyzed using commodity hardware.

A central question in social network analysis is the study of the community structure of social graphs. Several models have been introduced to capture the community structure of social networks [25, 28, 30] and several empirical studies describe community structures at a macroscopic level [20, 31, 30]. Among those works, the one of Leskovec et al. [30] stands out for its rich and deep analysis. The main finding of the paper is the lack of a clear macroscopic community structure in online social networks. More precisely, the authors show that it is very rare to observe medium size communities of small conductance.

Those results have been confirmed by recent studies that observed that real life communities hardly follow the assumptions of the most commonly used algorithms [1]. They strongly overlap with each other and often have more connections outside than inside themselves. This makes it hard to even define communities at the global graph level.

In contrast with these findings, it has been observed that while communities might be difficult to find at a global level, their structure seems to be much more clear at a microscopic level [12]. This is particularly true at the level of node-centric structures called ego-networks (or ego-nets) which are the subgraphs representing the connections among the neighbors of a given node.

Intuitively, this happens because, while nodes participate in many communities, usually there is a single (or a limited number of) context in which two specific neighbors interact. Nevertheless, to the best of our knowledge, no extensive study of the ego-net community structure has been performed on different large-scale datasets to confirm these observations.

In this paper, we argue that it is possible to address important graph mining tasks by analyzing the ego-nets of a social network and performing independent computations on them. In particular, we study extensively, both through experimentation on large-scale graphs and formally, the properties of such ego-net community structure and its possible application to the important problem of friend suggestion.

More precisely, our first contribution is to verify this intuition through an extensive analysis of the community structure of the ego-networks of several large-scale graphs (including an anonymized snapshot of the entire public Google+ graph). Thanks to this analysis we show that even if communities are hard to detect at a global level, they are easily detectable at a microscopic level.

To further validate this finding, our second contribution is to show the first large-scale study on on-line *social circle detection* showing that simple clustering algorithms, based solely on the graph structure, are able to match user-defined communities with great accuracy. Notice that while previous studies have only been applied to graphs with hundred of nodes we test the algorithms on the entire Google+ graph with hundred of millions of nodes.

Toward this end, our third contribution is the introduction

[*]Work partially done while intern at Google, Mountain View. Work partially supported by the NSF Award IIS-1247581; MIUR-PRIN National Project ARS TechnoMedia 2010N5K7EB (Algorithmics for Social Technological Networks); and Google Focused Award.

Proceedings of the VLDB Endowment, Vol. 9, No. 4
Copyright 2015 VLDB Endowment 2150-8097/15/12.

of a simple and efficient algorithmic framework for the parallelization of community detection algorithms on ego-nets in MapReduce for which we provide theoretical guarantees on its scalability[1].

We then turn our attention to possible applications of ego-net mining. Our forth and the most important empirical contribution is the definition of novel, and easily computable, similarity measures among nodes in a graph, which we name *ego-network friendship scores*. These scores are based solely on the analysis of the community structure of the ego-nets and, despite its simplicity and the fact that its computation involves only information in the immediate neighborhood of a node, we show experimentally with real data that it is able to predict link formation with good accuracy. The simplest such feature counts the number of ego-net clusters that two nodes participates in. Other features also take into account the quality of those clusters or the relevance of each node for each cluster. Via an empirical evaluation on a large real-world Google+ data set, first observe that even the simplest feature of these features outperforms all the most commonly used similarity measures based on the immediate neighborhood of a node. In order to gather an understanding of these phenomena, we also introduce a simple model of random networks for which we show formally the accuracy of this simple measure. Finally, we confirm our result by running a live experiment on Google+ using several features based on the ego-network scores. Our live experiment shows that adding such features improves the performance of the friend suggestion system in production. In particular, it improves the the probability of accepting the suggestion by 1%, and it decreases the probability of explicitly rejecting the suggestion by 3%.

The rest of the paper is organized as follows. In Section 2 we briefly review the literature relevant to our work. Then in Section 3 we define formally the ego-nets and some useful community quality metrics. In Section 4 we present our framework and the community detection algorithms we used in our experiments. Then in Section 5 we presents our findings on the properties of the ego-nets in our datasets. Later, in Section 6 we present our new friend suggestion technique and its results. Finally, in Section 7 we draw our conclusions and point at possible future directions.

2. RELATED WORKS

The definition of ego network as a central concept in sociology dates back to the seminal work of Freeman [19]. After their introduction, the analysis of ego networks established itself at the basis of social network analysis [10, 15, 18, 46].

Recently, great attention has been devoted in the computer science community on mining ego-nets. In their pioneering work, Rees and Gallagher [37] proposed to the use of ego-networks to find a global clustering of the graph. The core idea of their algorithm is to find basic communities by computing the weakly connected components for each ego-network after removing the ego from it. Then to obtain a global clustering they merge communities that overlap significantly. Coscia et al. [12] built on this work employing label propagation algorithms to cluster the ego-nets and analyzing different merging strategies.

In a different line of works, McAuley and Leskovec [35] provided a machine learning approach to cluster ego-nets using both the graphs and the attributes of the nodes. Subsequently Yang et al. [48] proposed an extension of this model for directed and undirected graphs. Finally Li et al. [32] extended their learning model to capture also hidden attributes that are not explicitly present in the input.

Thanks to the recent availability of user defined communities, such as Google+ circle [35] or Facebook friend list [35], some recent papers focused on the analysis of their structural properties. Steffen and Braurer [9] analyzed circles in Google+ and observed that they are more densely connected with the outside of the circle than normal "communities". Indeed they observe in their studies that most of the circles have conductance > 0.90. La Gala et al. [26], instead, defined new metrics for the strength of a ties in an ego-nets and also tested the Dunbar circles' hypothesis [15]. Finally, user defined social circles have also been used to design community detection algorithm by Hemank and Ramasuri [27] that clustered together nodes that are assigned to the same circle by many common neighbors.

Ego-nets have also been used to design various machine learning algorithm to solve several problems. Gleich and Seshadhri [21] noted the ego networks are good seed for the Personalize PageRank clustering. Akoglu et al. [3] showed that simple statistics on ego-nets help find suspicious and spam nodes. Wang and Li [45] employed ego-nets for mining citation networks. Finally, confirming the importance of ego-nets Sharma et al. [41] showed that suggestions based solely on information on the neighbors of node give performance in line (and sometimes superior) to using the entire graph.

Related topics to ego-nets includes the center-piece subgraph [44] notion: the subgraph of the most connected nodes to an arbitrary set of query nodes Q. This extends the definition of ego-nets which consist of the subgraph of nodes directly connected to a single node (the ego). Also related to ego-nets are local community [11, 4]—subgraphs in the vicinity of a node optimizing a certain quality function—and the viewpoint neighborhoods [6]—the set of nodes of most direct interest to a center node.

Another related topic is that of friend suggestion algorithms. The literature in this area is gigantic, and for this reason we focus here only on the most related results. In a classic paper in social network analysis [2] Adamic and Adar defined a first measure of similarity between not adjacent nodes in a social network. In a subsequent work [33] Liben-Nowell and Kleinberg compared several notions of similarity between nodes for friend suggestion. In [23] Gupta et al. proposed a friend suggestion algorithm for Twitter based on a combination of top-k Personalized PageRank ranking and a SALSA-like hubs and authority algorithm. Finally, Schall [39] analyzed different features that are useful for friend suggestion in directed graphs.

3. PRELIMINARIES

Let $G(V, E)$ be a (possibly directed) graph. Let $N(u)$ be the set of neighbors of u in the graph and $d(u) = |N(u)|$. When we work with directed graph we use $N^+(u)$ $(N^-(u))$ and $d^+(u)$ $(d^-(u))$ to refer respectively to the out-neighborhood

[1]Note that a naïve algorithm would take time $\Omega(\sum_{u \in V} d(u)^2)$ only for building the ego-nets. For this reason it is fundamental to design efficient algorithms to build and cluster egonetworks.

(in-neighborhood) of node u and its out-degree (in-degree), we also define $N(u) = N^+(u) \cup N^-(u)$ and $d(u) = d^+(u) + d^-(u)$.

We define the ego-net for a node $u \in V$ as the graph induced on G by $u \cup N(u)$ ($u \cup N^+(u)$ for direct graph). In the rest of the paper we denote the ego-net of a node u as S_u. Often we are interested in analyzing the properties of the graph represented by the ego-net of node u without the *ego* node u itself. The *ego-net without ego* graph of u is defined as the graph induced on $N(u)$ ($N^+(u)$ for direct graph), in the rest of the paper we denote the ego-net without ego of a node u as Z_u.

In the paper we make extensive use of community detection algorithms (a.k.a clustering algorithms). In order to evaluate the quality of a cluster we will look at two well known metrics: density [22] and conductance [24]. For completeness, here we recall their definitions.

Definition 1 (Conductance). *Let $G(V, E)$ be a graph and let $W \subseteq V$. The conductance $\phi(W)$ of W is defined as: $\phi(W) = \frac{|cut(W, V \setminus W)|}{Vol(W)}$, where $cut(X, Y)$ is equal to the set of edges between X and Y and $Vol(X) = \sum_{v \in X} d(v)$.*

Furthermore, we define the conductance of a clustering $\mathscr{C} = (C_1, C_2, \dots, C_k)$ as: $\Phi(\mathscr{C}) = \max_i \phi(C_i)$.

Definition 2 (Density). *Let $G(V, E)$ be a graph and let $W \subseteq V$, we define density of W and we denote it as $\psi(W)$ as: $\psi(W) = \frac{Vol(E)}{|W|(|W|-1)}$. Similarly, the density of clustering $\mathscr{C} = (C_1, C_2, \dots, C_k)$ is defined as: $\Psi(\mathscr{C}) = \min_i \phi(C_i)$.*

4. ALGORITHMS

In this section we introduce our algorithmic framework for the study of ego-nets' clusters in large-scale networks. The main challenge that we address is in efficiently computing a clustering of the ego-nets in communities for all nodes in the graph in parallel. We will define an efficient technique, implementable in MapReduce, that is applicable to any clustering algorithm.

The main challenge in this step is that a nave algorithm would take time $\Omega(\sum_{u \in V} d(u)^2)$ only for constructing the ego-networks. Furthermore this is true even when we add a threshold on the maximum size of the ego-nets that are analyzed. In fact, also in this case, high degree nodes would send their neighborhoods to all their neighbors to let them build their ego-nets. For this reason we need to develop efficient techniques to build and cluster them.

In the section we first describe briefly the clustering algorithms that we use to cluster ego-nets. Then we present our new technique to compute the clustering of all the ego-networks in a graph efficiently. Finally we discuss how to parallelize this technique in MapReduce in order to scale to very large networks.

4.1 Clustering algorithms

In the community detection literature, several algorithms have been introduced [31] to cluster social networks. In this paper, we mainly focus on five algorithms: *Personalized PageRank partitioning*, *Personalized PageRank overlapping clustering*, *Label Propagation clustering using Absolute Potts Model*, *Overlapping Label Propagation Clustering using SLPA*, *Hierarchical Clustering using k-cores.* We give a brief description of the algorithms in Appendix A.

4.2 Efficient ego-network clustering

In this subsection, we define our framework that allows efficient computation of ego-nets' clusterings for all nodes in a graph, using an arbitrary community detection algorithm. The basic idea is to use a technique similar to Schank's algorithm [40] for triangle counting and its parallelization [42].

Suppose that a clustering algorithm runs on a graph with m edges in time $t(m)$. A trivial approach to this problem would be to first construct, for all nodes $v \in V$, the ego-nets of v by exploring the adjacency list of each node $w \in N(v)$ and by computing $N(w) \cap N(v)$. Then we will apply the clustering algorithm on the ego-net of v.

Observe that this procedure can be implemented in time $\sum_{v \in V} \left(t(m_v) + \sum_{w \in N^+(v)} (d(w)^2) \right)$, where m_v is the number of edges in the ego-net of v. This can be bounded to $O\left(n \cdot t(\Delta^2) + n^3\right)$, where Δ is the maximum degree in the graph and n is the number of nodes. Clearly this technique does not scale in large graphs (not even using parallelization) as nodes might have very high degrees.

Efficient ego-network computation. Our first algorithmic contribution is to design a technique to compute all the ego-networks in a graph in time $O\left(m^{3/2}\right)$. The main observation is that $(v, z) \in N(u)$ is an edge in the ego-net S_u if and only if u, v, z form a triangle in $G(V, E)$. We can then use for our problem a simple variation of the efficient Schank's algorithm that enumerates all triangles in a graph. We first recall Schank's algorithm and then show how to adapt it to compute ego-network.

The intuition behind Schank's algorithm is to look sequentially at triangles around a minimal degree node and then by deleting the already analyzed nodes (see Algorithm 1 for a pseudo-code).

Algorithm 1 Schank's algorithm for enumerating triangles

```
Input: G(V,E)
Output: All triangle of $G$
while $V \neq \emptyset$ do
    $u \to$ node of minimal degree.
    for $\forall v, z \in N(u)$ do
        if $(v, z) \in E$ then
            Output triangle $u, v, z$
        end if
    end for
    Delete node $u$ and its adjacent edges.
end while
```

Schank [40] showed that such algorithm requires $O\left(m^{3/2}\right)$ time. Based on this result and our observation above, we design a fast algorithm to compute the ego-nets. The pseudocode of this algorithm is described in Algorithm 2.

As corollary of the Schank result we have:

Corollary 1. *It is possible to construct all the ego-networks in a graph in time $O\left(m^{3/2}\right)$.*

Bounding the clustering running time. Note that by Corollary 1 we also obtain that the total number of edges over all the ego-networks is $O\left(m^{3/2}\right)$. Assume that the running time of a clustering algorithm on a subset S, $t(\cdot)$, is a

Algorithm 2 Fast ego-network construction

```
Input: G(V,E)
Output: All ego-nets of nodes of $G$.
while $V \neq \emptyset$ do
    $u \rightarrow$ node of minimal degree.
    for $\forall v, z \in N(u)$ do
        if $(v, z) \in E$ then
            Add $(u, v)$ to $S_z$
            Add $(v, z)$ to $S_u$
            Add $(u, z)$ to $S_v$
        end if
    end for
    Delete node $u$ and its adjacent edges.
end while
```

convex function in the number of the edges in S.[2]. Using the previous observations and a basic fact on convex functions we can obtain a simple bound on the total running time of the clustering algorithm applied to all ego-nets.

FACT 1. *Let f be a convex function, and let $x_1, x_2, \ldots, x_n$ be such that $\sum_i x_i = n$ and $\max_i x_i = C$, then f is maximized when $\lfloor \frac{n}{C} \rfloor$ variables are set to C, one variable is set to $(n \mod C)$ and the remaining variables are set to 0.*

PROOF. Suppose that this is not true, it means that there are 2 variables x_i, x_j with $x_i, x_j \neq 0$ and $x_i, x_j < C$. Assume without loss of generality that $x_i > x_j$, then we will prove that $f((1+\epsilon)x_i) + f((1-\epsilon)x_j) \geq f(x_i) + f(x_j)$. Note that by definition of convex function $\frac{f((1+\epsilon)x_i) - f(x_i)}{\epsilon} \geq \frac{f(x_j) - f((1-\epsilon)x_j)}{\epsilon}$. □

Suppose that $\mathcal{A}$ is any graph clustering algorithm with time complexity $t(|E_S|)$ where t is convex function of the size of the set E_S edges in subgraph S in input. As the number of edges in an ego network is $O(m)$ and by Fact 1 we can prove the following properties.

COROLLARY 2. *The running time of algorithm $\mathcal{A}$ on all the ego-networks is $O(\sqrt{m} \cdot t(m))$.*

Combining the two Corollaries we get the following lemma:

LEMMA 1. *The total running time to compute the ego-networks and to run the clustering algorithm $\mathcal{A}$ on all of them is $O(\sqrt{m}t(m) + m^{3/2})$.*

Parallel implementation. The previous results show how to implement ego-network clustering efficiently in a sequential setting. To parallelize the algorithm in the MapReduce framework we employ similar techniques to the ones used in [42] to parallelize Schank's algorithm.

Before giving the pseudo-code for our algorithm we briefly recall the main aspect of the MapReduce framework [13]. In the MapReduce framework the input/output data is represented by a pairs $\langle key, value \rangle$. In each round of MapReduce a Map and a Reduce functions are executed in parallel on several machine. In the Map phase each $\langle key, value \rangle$ is assigned to an arbitrary mapper and the Map function is executed on them. The map function's output is a set of new pairs $\langle key', value' \rangle$. In the reducer phase all the pair having the same key are analyzed by the same Reduce function.

The parallel Shank's algorithm first partitions the nodes in the graph in $\rho \in \Omega(\sqrt{n})$ random partitions $V_1, V_2, \ldots V_\rho$. Then instead of counting the triangle in the whole graph at the same time, it enumerates them in parallel for all the graph $G(V_i \cup V_j \cup V_z, E_{V_i \cup V_j \cup V_z})$, $\forall 0 < i, j, z \leq \rho$ and $i \neq j \neq z$.

Using similar techniques we can define our algorithm for efficiently computing ego-nets in parallel (see Algorithm 3 for the pseudo-code).

Algorithm 3 Fast parallel ego-network construction

```
Map: Input: edge $(u, v)$
{Let $h(\cdot)$ be a universal hash function into $[0, \rho]$}
$i \leftarrow \lceil h(u) \rceil$
$j \leftarrow \lceil h(v) \rceil$
if $i == j$ then
    for $z \in \{1, 2, \ldots, \rho\} \wedge z \neq i$ do
        for $w \in \{1, 2, \ldots, \rho\} \wedge w \neq i, z$ do
            Output $(sorted(i, z, w), (u, v))$
        end for
    end for
else
    for $z \in \{1, 2, \ldots, \rho\} \wedge z \neq i, j$ do
        Output $(sorted(i, j, z), (u, v))$
    end for
end if
Reduce: Run Algorithm 2 on the input graph
```

Note that after constructing the ego-nets we can run in parallel on each of them our clustering algorithm $\mathcal{A}$ in another step of MapReduce.

As a corollary of the results in [42] and Lemma 1 we can bound the total parallel work for our framework algorithm:

LEMMA 2. *Then the total amount of parallel work to compute the ego-nets and to run the clustering algorithm $\mathcal{A}$ on them is $O(\sqrt{m}t(m) + m^{3/2})$ and the algorithm executes only 2 MapReduce iterations.*

Very high degree nodes. Notice that while our techniques significantly improves the scalability of ego-net mining in graphs with high degree nodes (avoing the quadratic running time of the naive algorithm), still the ego-nets of highly popular nodes might be large to store and analyze. Indeed, in the worst case, for a node connected to the entire network, the ego-net is as large as the entire graph. We observe that this is not, however, a fundamental issue for many of the applications of ego-net mining. The ability of people to maintain significant social ties with a large number of contacts is constrained by the so-called cognitive capacity (as witnessed by the celebrated Dunbar's number theory [14, 30]). So we can expect that the study of extremely large ego-nets to be less relevant.

Experiments. Our experiments confirm the theoretical insight on the speedup achieved by the algorithm presented. For graphs of moderate size we compared the execution of the naive sequential $O(\sum_u \deg^2(u))$ algorithm with that of the fast sequential algorithm based on Schank's technique (Algorithm 2) as well as our distributed algorithm (Algorithm 3). In our Livejournal dataset, for instance, the fast sequential algorithm has a speedup of 5x (w.r.t. to the naive

[2]Note that is true for all the algorithm presented in the paper and in general for any "reasonable" community detection algorithm.

algorithm) while the distributed algorithm has a speedup of 11x.

5. REAL-WORLD EGO-NET PROPERTIES

In this section, our main focus is to understand the statistical properties of the community structure of ego-nets in real-world graphs. To this end, we apply our previously defined algorithms to efficiently compute (and cluster) the ego-nets of several large-scale graphs, including the entire Google+ network. In particular we focus on the following question: do ego-networks have a good clustering?

To study this topic in depth we first analyze the clustering of the ego-networks of 8 publicly available datasets: Amazon co-purchase network, Astro Physics collaboration network, Enron email network, Slashdot social network, patents citation network, Facebook social network, LiveJournal social network and Twitter social network. In this version of the paper we present results only for Patents, Facebook, LiveJournal, Twitter and Google+ in interest of space, the results for the other networks are very similar. The datasets are described in Table 1.

Graph	$\|V\|$	$\|E\|$
Patents [29]	3774768	16518948
Facebook [35]	4039	88234
LiveJournal [7]	4847571	68993773
Twitter [35]	81306	1768149
Google+	XXXM	XXXM

Table 1: Number of nodes and edges for graphs analyzed in the experiment. Exact figures for the proprietary Google+ graph are not disclosed.

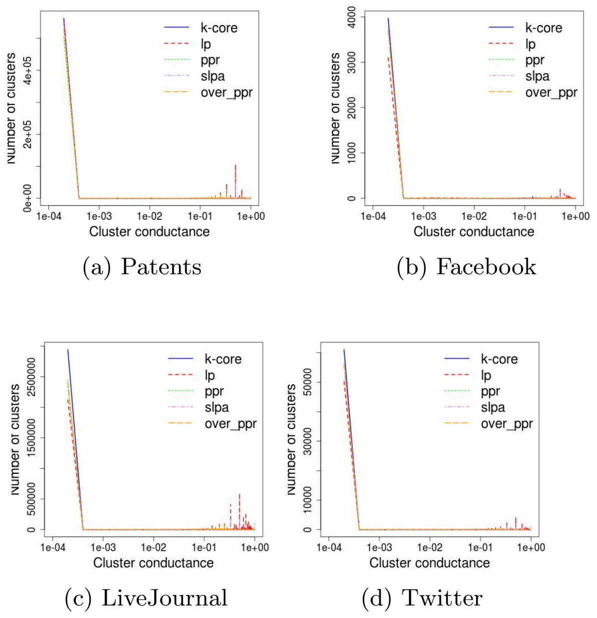
Then we compare the clusters retrieved by our algorithms with user defined circles. For this purpose, we analyze a snapshot of the entire public Google+ network containing hundreds of millions of users and edges with the complete list of public circles. In this analysis, we also describe some interesting properties of Google+ circles.

5.1 Egonetwork community structure

In this subsection, we analyze the clustering structure of the ego-networks of our publicly available datasets. We start by looking at the conductance of all the clustering obtained by our 5 clustering algorithms, the results of this analysis are shown in Figure 1 (note that the x axis is in log scale).

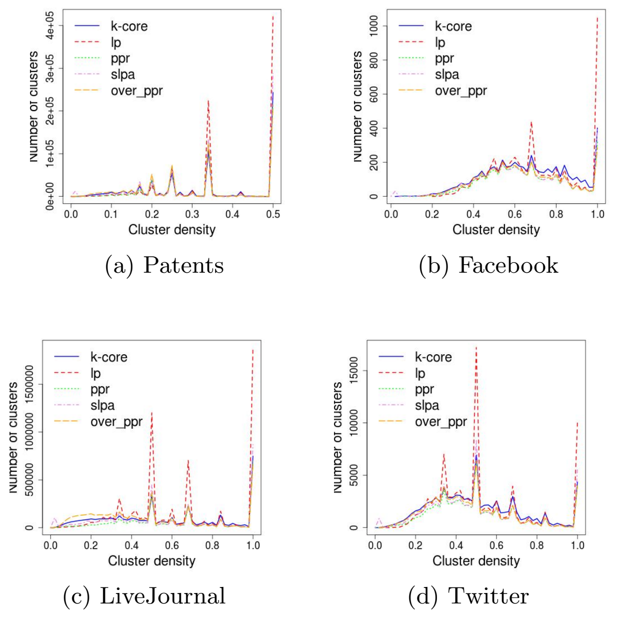
There are a few interesting results to observe. First, overall all the clustering algorithms are able to find clusters with very low conductance. In particular most of the clusters retrieved by all the algorithms have conductance close to 0 or actually zero. Second, the k-core algorithm and the label propagation algorithm (lp) generate a larger number of clusters. This is not surprising because the lp algorithm is designed to find small communities and the k-cores can be numerous for small k value. Nevertheless it is interesting to note that the quality of the communities is still generally high. It is also interesting to note that the PPR-based algorithms find communities with lower conductance, as expected, while the label propagation one find several clusters with larger conductance (for example look at Figure 1(c)).

Now we turn our attention to Figure 2 where we analyze the density of the clusters. The results that we obtain in this case are less homogeneous than the one that we had for the conductance. In particular, we observe that in social graphs



(a) Patents (b) Facebook (c) LiveJournal (d) Twitter

Figure 1: Number of clusters with a specific conductance.

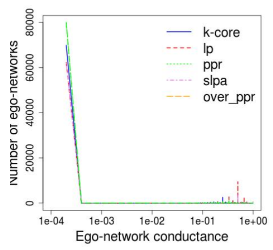


(a) Patents (b) Facebook (c) LiveJournal (d) Twitter

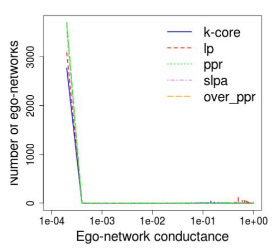
Figure 2: Number of clusters with a specific density.

as Facebook, Livejournal and Twitter (Figures 2(b), 2(c) and 2(d), respectively) few clusters have very high density. Nevertheless for all the graphs and for all the clustering algorithms most of the retrieved clusters have density larger than 0.2 which is particular surprising, as it implies that in the graph induced by the most of the retrieved clusters there is at least one edge every 5 possible edges, so those clusters look almost like a clique.

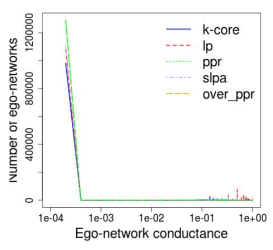
A possible explanation for those results is that many of the



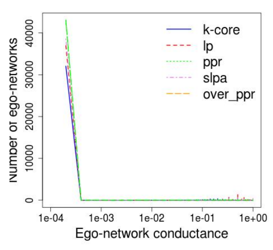
(a) Patents



(b) Facebook



(c) LiveJournal



(d) Twitter

Figure 3: Number of egonets with a specific conductance.

retrieved clusters are of small size and not really meaningful. To address this question, we consider the same conductance and density metric at ego-network level and not at clustering level. As presented in Section 3, we define the conductance of a ego-network to be equal to the maximum conductance of a cluster in the ego-network clustering. For the density of a clustering we use the min density.

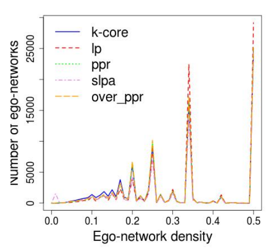
Interestingly, even by analyzing our clustering at this level of granularity the results do not change significantly. In fact, looking at the plots describing the ego-network conductance (Figure 3), it is possible to observe that the conductance of the retrieved clusters is typically very low and that the PPR-based algorithms find clusters with lower conductance than the other and that label propagation algorithms yields slightly higher conductance clustering.

Finally, we turn our attention to the density of the clusters at the ego-net level (Figure 4). Also in this case, the results are similar to the one observed at the cluster level. The clusters have typically very high density and the label propagation algorithm retrieves clusters with slightly higher conductance.

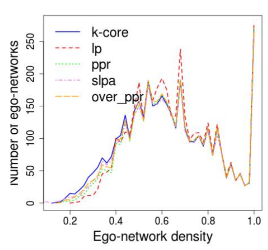
It is interesting to note that at ego-net level the k-core algorithm finds clusters of low density. This may look surprising at first sight, but it is not because the k-core algorithm enumerates all k-core clusters also for small k. So it is not surprising that the cluster of minimum density in an ego-network has actually a low density.

5.2 Egonetwork communities vs circles

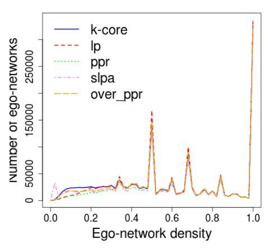
Now we turn our attention to a comparison between the communities that we retrieve using our clustering algorithms and the user defined circles. Intuitively user defined circles, if available and updated, are good clustering of the ego-



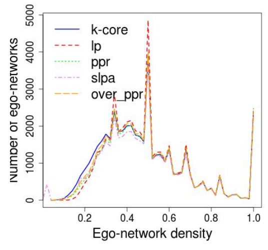
(a) Patents



(b) Facebook



(c) LiveJournal



(d) Twitter

Figure 4: Number of egonets with a specific density.

nets. For this reason we compare our clustering with them[3]. For our analysis we use the entire Google+ public graph for which we possess hundreds of millions of user-defined circles. Note that our analysis is the first analysis on circles involving hundreds of millions of nodes and their circles.

First, we look at some basic statistics on circles properties. We briefly summarize the results in the following Table 2.

Metric	Values
Average density	0.352
Average conductance	0.1438
Average number of wcc	4.981
Average fraction of number in largest wcc	0.726

Table 2: Basic metrics for several circles properties.

Algorithm	N. Pr.	N. Rc.	E. Pr.	E. Rc.
k-core	0.77	0.64	0.80	1.0
lp	0.86	0.49	0.87	0.96
ppr	0.86	0.48	0.87	0.93
over. ppr	0.84	0.55	0.86	0.94
slpa	0.85	0.53	0.86	0.98

Table 3: Overlap between circles and clusters retrieved by different clustering algorithms. N. and E. stand for node and edge, respectively, while Pr., Rc. stand for Precision, Recall respectively.

For each circle we compute its density, conductance, number of weakly connected components and the fraction of nodes in the largest weakly connected component. Previous findings focused on the macroscopic scale of the graph [9] observed high conductance in Google+ circles making them difficult to separate with clustering algorithm. In contrast to this study we focus on the ego-net level scale where we

[3]Note that our goal here is not to retrieve circles but just to evaluate the quality of our clustering from a different prospective

observe that user defined clusters have typically low conductance and high density so they resemble the classic notion of clusters.

We compared user defined circles with the communities retrieved by our clustering algorithms. For this analysis we use as ground-truth communities the connected components of each circles. In particular we study how different clustering algorithms generate clusters that resemble user defined ground-truth circles. Before describing our results, we introduce four metrics that we will consider.

Node-Pair Precision/Recall The node-pair error rate computation is obtained by modeling the problem of reconstructing circles as an information retrieval problem. We define the set of pairs of nodes in the same circle, $U = \{(x, y) | x, y \in C\}$, as the set of items to retrieve. Then the set of pairs of node that ends up in the same cluster is considered to be the set of items retrieved. This allows to evaluate the accuracy of the clustering using all the well-known information-retrieval measures of precision and recall.

Edge Precision/Recall This measure is similar to the previous one but instead of using the set of pairs of nodes in the same cluster as ground-truth, we use the set of edges inside a circle.

The results are reported in Table 3. It is interesting to notice that despite the simplicity (and scalability) of the approach all the algorithms are quite good in retrieving user defined circles. In fact they all have precision $> 80\%$ and recall at least 60%. We observe that *k-core* algorithm has highest score considering Node Recall while *slpa* outperforms the other algorithms for Edge scores[4]. Our experiments on friend suggestion, reported in Section 6, will confirm these two algorithms to be among the best performing ones also for that task.

6. FRIEND SUGGESTION VIA EGONETS CLUSTERING

In the previous section, we have observed that ego-net clusters obtained by community detection methods, closely match user-defined circles. We now study a practical application of ego-networks clustering to the important task of friend suggestion. As a driver of social engagement, friend suggestion is a fundamental tool for online social networks. Designing a high quality system is very challenging and the state of the art friend suggestions tools are based on a multitude of features combined by efficient machine learning algorithms.

A very effective family of features for friend suggestion is the set of graph based features. In this family, it is possible to distinguish between two main groups: (i) Features computed on the entire structure of the graph [35, 48], (ii) Features computed locally by analyzing the neighborhood of a node [2, 33]. In the rest of this section we show theoretically and experimentally that ego-networks clustering can be used to produce a new local network feature that outperform all the previous network features and that is impactful in real world experiments.

One of the most intuitive and commonly used feature for friend suggestion is the number of common friends between two users. This feature is easy to compute even for very large graph and it has a very intuitive meaning. Nevertheless it has also few limitations, high degree nodes generate a lot of suggestions even if they have only "weak" links. Second this feature tends to suggest more often high degree nodes. Finally a user is typically member of several social groups and this feature tends to suggest unrelated contacts as it ignores the social groups to which these common friendships belong.[5]

To provide an intuition for the latter problem consider the following real-life example. Users A and B have 4 common friends X_1, X_2, X_3, X_4 while A and C have only 3 common neighbors Y_1, Y_2, Y_3. Although from the point of view of X_i's A and B belong to different social circles: for instance A is a family member of X_1 while B is instead a co-worker of X_1, etc. On the other hand for all Y_i's A and B belong to the same circle of Y_i (e.g. they are both co-workers of Y_1, family member of Y_2, etc.). In this case it is more likely that A and C are friends w.r.t. to A and B despite having fewer common neighbors.

To solve the problem of high degree nodes several techniques have been proposed, Adamic and Adar [2] proposed to score a suggestion with weight inversely proportional to the degree of the common friend between two nodes. The second problem can be solved by considering the Jaccard coefficient between the neighborhood of two users. Nevertheless the third problem seems more complex and to the best of our knowledge no formal technique has been developed to solve these issues.

In the following, we show how to tackle these issues via clustering ego-nets. The main idea is to cluster Z_u, the ego-network minus ego of user u, in social groups so that then we suggest two users if and only if they share a cluster in some ego-net clustering. In particular, the simplest *ego-network friendship score* between two nodes v and w can be defined as: $W(v, w) = \sum_{u: v, w \in N(u)} \begin{cases} 1 & \text{if } v, w \text{ in the same cluster of } Z_u, \\ 0 & \text{otherwise.} \end{cases}$

Essentially, for a target node v, $W(v, w)$ values the suggestion of nodes w (to node v) as the number of joint friends u of v and w s.t. from the point of view of u's ego-net both v and w belong to the same sub-community of the friends of u. The social intuition is that pairs of nodes v, w that belongs to the same ego-net sub-community of many common friends have higher probability of forming a link (indeed one of these common friend might have introduced them to each other).

Note that this score can be generalized to take into account other properties of the communities to which v, w belong in the same ego-net, such as the degrees of v and w, the density of the cluster, the fraction of edges of u or v in the cluster and other features as we do in our live experiments. However, for the sake of comparing to previously studied features, we mainly compare with the above simple score.

Notice also that this measure can be computed easily in a single MapReduce step. In fact, each ego-net cluster produced in the previous phase can be analyzed independently of the others, in parallel, thus enabling the scalability of our methods to very large datasets.

Finally, this score naturally discounts the importance of celebrity nodes for two reasons. First, we define the ego-nets as the out-neighborhood of a node. Hence, the suggestions

[4]It is interesting to note that also when we restrict our analysis to circles of size at least 10 we obtain similar results.

[5]Note that this is an issue also for methods that ensembles path of length bigger than one between the user and a possible suggestion.

induced by a celebrity ego-net will not affect the large number of followers of the celebrity (the in-neighbors) but instead on only the significantly smaller number of users added by the celebrity in his/her circles (they out-neighbors).[6] Second, while celebrity nodes will be present in many ego-nets, they will only be suggested to a user u, if they are clustered in the same community of user u in many ego-nets of friends of u. Arguably, in this case, the celebrity could be a good suggestion.

6.1 A simple theoretical model

To prove the strength of our new feature, we introduce a new simple theoretical model in which we formally prove that the simple *ego-network friendship score* defined above performs better than the number of common friends and the Adamic Adar score. The model that we analyze here is simplistic (at the end of this section we comment on its limitations), but it allows us to mathematically formalize the intuition behind the interesting properties of our new score. Our model has similarities with the models in [5, 8].

In our model, each of n people has a set of interest $\mathscr{I}$ of size $k > 1$ (possibly with repetitions), where k is the same for all nodes in the graph. We assume that k interests of a user are selected uniformly at random (with replacement) from a set of $\frac{n}{\log^2 n}$ possible interests. Given these interest relationships, the social graph is formed by creating an edge between two people with probability $p > 0$ if they share at least one interest. Note that our graph is built by composing two types random graphs, the first one is a bipartite random graph between people and interest(an affiliation network) and the second one is a random graph defining friendships within any pair of interest.

Given a social graph G as input, we want to generate good friend suggestions without knowing the interest set for each person. We define a friend suggestion to be good or valid if the two people involved in the suggestion share at least one interest. We define a suggestion bad otherwise.

It is possible to show that in this model, the number of common friends similarity score and the Adamic Adar score provides the same set of friend suggestions containing a similar fraction of good and bad suggestions. In particular we show that the set of good suggestions J^+ is of the same order of magnitude that the set of bad suggestions J^- provided by this measure (hence we have an overall poor performance). On the other hand, we show that w.h.p.[7] the *ego-network friendship score* using a very simple clustering algorithm returns all the good suggestions in J^+ with a significantly small number $o(|J^+|)$ bad suggestions. So *ego-network friendship score* provably outperforms the other techniques.

For simplicity, the clustering algorithm that we consider in the proofs in this section is the simple connected component algorithm. This may look a bit artificial, however, in the next section we will show also experimentally, with real data, that this simple clustering algorithm (applied in our ego-net based similarity score) outperforms the number of common friends and the Adamic Adar score. Nevertheless, we also observe that in practice it is possible to obtain better results by using more sophisticated clustering algorithms and we cojecture that our theoretical result could be extended to many other algorithms.

In order to show our theoretical results, we first prove some properties of our model. First we find a bound on the size of any given community.

Fact 2. *Let ϵ be any constant $\epsilon > 0$ for any interest i, the number of people interested to a specific interest i, V_i is $(1-\epsilon)k\log^2 n < |V_i| < (1+\epsilon)k\log^2 n$ with probability $1 - O(n^{-\log n})$.*

Proof. Let $X_v(t)$ be the indicator variable that it is 1 if the t-th interest of v is i and 0 otherwise. Then $E[V_i] = \sum_{v\in V}\sum_{t=1}^{k} X_v(t) = nk\frac{\log^2 n}{n} = k\log^2 n$. Furthermore the $X_v(t)$ are i.i.d. random variables in $[0,1]$ so we can apply the Chernoff bound and obtain that $P(|V_i - E[V_i]| > \epsilon k \log^2 n) \leq O(n^{-\log n})$. Now there are in total $\frac{n}{\log^2 n}$ interests, so by applying the union bound we get the result. $\square$

Then we bound the number of people sharing more than one interest [8].

Fact 3. *The number of pairs of users that share more than one interest is w.h.p. $O(\log^5 n)$.*

Proof. Let $X_{u,v}$ be the indicator variable that is 1 if u and v share more than one interest. $E[X_{u,v}] = P(X_{u,v}) \leq \binom{k}{2}\left(\frac{k\log^2 n}{n}\right)^2 \in O\left(\log^4 n/n^2\right)$. So the total number of expected pairs, Y, that share more than one interest is $E[Y] \leq n^2 E[X_{u,v}] \in O\left(\log^4 n\right)$. So using the Markov inequality we get that the probability that the number of pairs of users that share more than one interest is $O(\log^5 n)$ is $1 - O\left(1/\log n\right)$. $\square$

Fact 4. *Let $G = G(\log^2 n, p)$ be a random Erdős-Rényi graph [17]. If $p > 0$ is constant, then the subgraph induced by the neighborhood of any node in G is connected with probability $1 - O(n^{-C\log n})$, for positive constant C.*

Proof. (Sketch) The core idea behind the proof is to show that, even if two neighbors of a node are not connected, there is a third neighbor connecting them.

Let v be any node in G, note that $N(v)$ is in expectation $p\log^2 n$ so using the Chernoff bound we have that the probability that $\frac{p}{2}\log^2 n \leq |N(v)| \leq 2p\log^2 n$ is $1 - O(n^{-\log n})$.

In the rest of the proof we assume $\frac{p}{2}\log^2 n \leq |N(v)| \leq 2p\log^2 n$. Pick an element z in $N(v)$, the expected size of $|N(z)\cap N(v)|$ is bigger or equal than $\frac{p^2}{2}\log^2 n$ so using the Chernoff bound again we have that the probability that $|N(z)\cap N(v)| \leq \frac{p^2}{4}\log^2 n$ is $O(n^{-\log n})$. So in the rest of the proof we assume that $|N(z)\cap N(v)| \geq \frac{p^2}{4}\log^2 n$ (Note that this happens with probability $1 - O(n^{-\log n})$).

Now consider any element $w \in N(v)\setminus N(z)$, we show that even if w is not directly connected to z there is a third neighbor in $N(v)\setminus N(z)$ that connects them. Note that the probability that w is not connected with any element in $N(z)\cap N(v)$ is $(1-p)^{|N(z)\cap N(v)|} \leq (1-p)^{\frac{p^2}{4}\log^2 n} \in O(n^{-p^2\log n})$. So by taking the union bound on the elements

[6]Many social networks caps the number of users that can be followed but instead the number of followers can be extremely high.

[7]We say that an event happens with high probability (w.h.p) it is happens with probability $1-o(1)$ for n number of nodes that goes to infinity.

[8]Note that the bounds that we present here are not tight, in our exposition we decide to trade some $\log n$ factors to simplify the proofs.

in $N(v) \setminus N(z)$ we have that all the elements in $N(v)$ are either connected to z or connected to it with a path of length 2 with probability $1 - O(n^{-p^2 \log n})$. $\square$

Using the previous three Facts, we can prove the main Lemma of this section (we note that similar lemma can be proved for Adamic-Adar score).

Lemma 3. *Let J^+ be the set of good suggestions and J^- be the set of bad suggestions for the friend suggestion problem produced by the number of common friend feature in our model. Then w.h.p. $|J^+| \in \Theta(|J^-|)$. Furthermore w.h.p. all the suggestions in J^+ are produced also by the ego-network friendship score but the number of bad suggestion for the ego-network friendship score is $o(|J^+|)$.*

Proof. (Sketch) From Fact 2 and by the Chernoff bound we know that every node has w.h.p. $O(\log^2 n)$ neighbors spitted in a constant number of communities of roughly equal size. Thus $|J^+|, |J^-| \in \Theta(\log^4 n)$ w.h.p.

Now consider *ego-network friendship score* by Fact 4 we know that w.h.p. for every node u, the nodes $N(u)$ that share an interest are connected w.h.p. so if we run connected components on all the ego networks we recover all the good suggestions. So w.h.p. *ego-network friendship score* makes all the suggestions in J^+.

Furthermore if *ego-network friendship score* makes a bad suggestion when it analyzes a node v if and only if the connected component algorithm clusters together two groups of people that share different interests in $N(v)$. But this happens only if v share two interests with another node and by Fact 3 this happens w.h.p. only $O(\log^5 n)$ times. So w.h.p. the number of bad suggestions produced by *ego-network friendship score* is $o(J^+)$. $\square$

Limitation of our theoretical model. We note that our model has a few important limitations: the degree of the nodes in the graph is almost regular and also the number of people interested to each interest in the graph is almost regular. Those two assumption are not realistic, nevertheless we think that our model gives good intuitions on why the *ego-network friendship score* is an effective feature as we will show experimentally in the next subsection.

6.2 Experimental results

As our first experiment, we analyze the prediction power of the simplest *ego-network friendship score* on a real-world data set, and compare it with the number of common friends and the Adamic-Adar score. We analyze an anonymized snapshot of the entire public Google+ graph, and computed link suggestions using the number of common friends, the Adamic-Adar score and the *ego-network friendship score* computed with different clustering algorithms including connected components. Then we observe the evolution of the network in the following two weeks and we check how many of the newly added edges where predicted by the three score. The precision and recall plots for the experiment are presented in Figure 5, Figure 6 and Figure 7.

From the figures, it can be observed that the new simple *ego-network friendship score* is very powerful and outperforms significantly the previously defined features. Moreover, ego-net friendship score defined based on different clustering algorithms have comparable performance, and among all clustering techniques the ego-net friendship score which is computed based SPLA has the best performance among the clustering methods that we tried. LPA and k-core comes next showing surprisingly close performances to the simple C.C. algorithm. The variations of PPR rank lower but still significantly better than common neighbors or Adamic Adar. Notice that the relative order of the algorithms is similar to the ones observed in the reconstruction of the circles as reported in Section 5.2 where SPLA and k-core out-perfome the PPR-based algorithms.

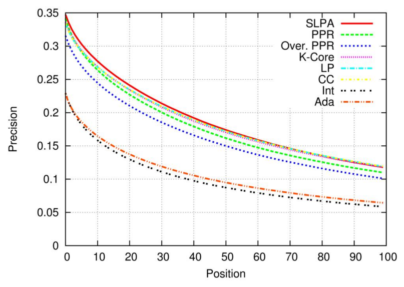


Figure 5: Precision of different features for friend suggestion at position i. Where the position is computed considering the descending feature score ordering. See Appendix for algorithms.

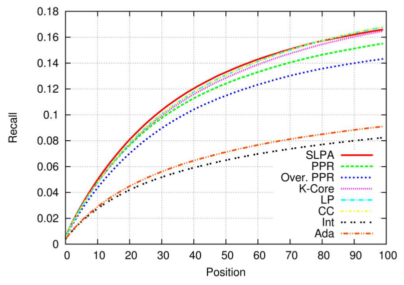


Figure 6: Recall of different features for friend suggestion at position i. Where the position is computed considering the descending feature score ordering. See Appendix for algorithms.

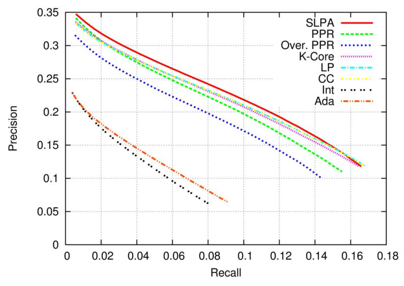


Figure 7: Precision/Recall plots of different features for friend suggestion. See Appendix for algorithms.

Real-world live experiment. Finally, we also run live experiments using ego-net-based features. In these live experiments, we went beyond the simple ego-net-based score and introduce other variants of this score. In particular, we add to the Google+ production model four features based on ego-network similarity:

$$W_1(v,w) = \sum_{\substack{u:v,w\in N(u),\\ v,w\in C,\\ C\subseteq Z_u}} 1 \qquad W_2(v,w) = \sum_{\substack{u:v,w\in N(u),\\ v,w\in C,\\ C\subseteq Z_u}} \psi(C)$$

$$W_3(v,w) = \sum_{\substack{u:v,w\in N(u),\\ v,w\in C,\\ C\subseteq Z_u}} |N_C(v)\cap N_C(w)|$$

$$W_4(v,w) = \sum_{\substack{u:v,w\in N(u),\\ v,w\in C,\\ C\subseteq Z_u}} \frac{\min\{|N_C(v)|,|N_C(w)|\}}{|C|}$$

where C is a cluster of Z_u, $\psi(C)$ and $N_C(u)$ are the density of the cluster C and the set of neighbor of u induced on C, respectively.

Note that $W_1(v,w)$ is the simple ego-network similarity we defined in the previous section. In addition to this simple score, we also consider $W_2(v,w)$ that takes into account the cluster density, $W_3(v,w)$ that takes in account the closeness of two nodes inside a cluster and $W_4(v,w)$ that takes in account the node connectivity inside clusters.

To better understand the differences between this four features we study the correlation between their values and the apply rate for the Google+ friend suggestion. In particular, we look at a week of Google+ data and for each past suggestion we compute its feature score using the 4 different scoring function. In Figure 8 we show the correlations between the scores and the apply rates.

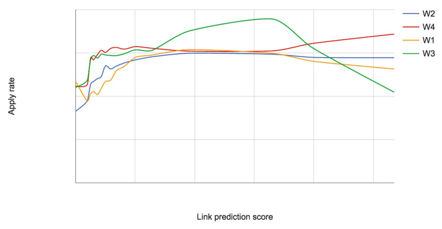


Figure 8: Correlation between feature scores and apply rates. Exact apply rates and feature scores are not disclosed.

It is possible to observe that all the features have positive correlation and that W_4 has stronger correlations.

Now we turn our attention to the performance of the new features in a live experiment with the Google+ friend suggestion system. In the experiment we added these 4 features to the Google+ production model which contains several semantic and network based features. Interestingly, even if the current production model is very complex and highly optimized, the ego-network similarity features had a clear positive impact on the prediction task.

On Google+, users receive a certain number of friend suggestions. A user may either accept a suggestion (adding a new user to his/her circles), explicitly reject a suggestion (which will not be shown by the system again) or simply not act on the suggestion (no user action is observed). We define the acceptance rate as the ratio of suggestions that are accepted, and the rejection rate as the ratio of explicitly rejected suggestions. We run a live experiment in G+ for 10 days (3/10/2015 - 3/20/2015) and after adding these new features to our supervised learning method we observed an increase in the acceptance rate by more than 0.5% and a decrease of the rejection rate by more than 1.4%. Furthermore those results are even stronger when we restrict our attention to more active user[9], in fact in this case we observed an increase in the acceptance rate by more than 1.5% and a decrease of the rejection rate by more than 3.3%. Note that this result in is inline with our previous theoretical and empirical results: friend suggestion score improves over the standard measures (like common neighbors) both in our theoretical model and in the off-line experiments.

7. CONCLUSIONS AND FUTURE WORKS

In this paper we focus on analyzing the structural properties of ego-networks. Our findings are quite interesting, ego-networks are easily clusterable and the user defined circles are somehow similar to the cluster retrieved by classic clustering algorithms. Toward this end we also developed an efficient technique to cluster all the ego-networks in a graph in parallel efficiently. Finally, we develop a new feature for friend suggestion, the *ego-network friendship score*, and prove theoretically and experimentally that our new feature outperforms the most well-known features that are based on the immediate neighborhood of a node.

We believe that many future directions of work might stem from our preliminary results. For instance, ego-networks mining and the study of ego-nets structure in bipartite graphs might be worth to be explored. Also, ego-net mining could be potentially used to detect spam requests to join circles. Moreover, better performances in friend suggestion could be obtained by combining the result of the application of different clustering methods.

Another possible direction is that of accounting for the dynamicity of nowadays social networks by extending our framework for ego-net mining to the context of dynamic graph streams. Conceivably, previous work on triangle counting in streams [34] might be beneficial to this direction (due to the strong connection between ego-net and triangles) as well as dynamic community detection algorithms [16].

Finally, while we showed that our scoring technique increases accuracy in friend suggestion by restricting recommendations within the community of the user, we recognize that this might prevent the user from exploring new interesting circles. This might result in a *social* filter bubble akin to that experienced in other recommendation systems [36]. An interesting future direction of work would be to explore the tradeoffs between exploitation of current community structure and exploration of further circles, in order to provide better and more comprehensive suggestions to the users.

Acknowledgements

We thank Chayant Tantipathananandh for his help and Ed Chi for the useful discussions.

8. REFERENCES

[1] B. D. Abrahao, S. Soundarajan, J. E. Hopcroft, and R. D. Kleinberg. A separability framework for analyzing community structure. TKDD, 2014.

[9]The set of active user is defined as the set of users that take 20 actions per day.

[2] L. A. Adamic and E. Adar. Friends and neighbors on the web. Social Networks, 2003.

[3] L. Akoglu, M. McGlohon, and C. Faloutsos. oddball: Spotting anomalies in weighted graphs. In PAKDD 2010, 2010.

[4] R. Andersen, F. R. K. Chung, and K. J. Lang. Using pagerank to locally partition a graph. Internet Mathematics, 2007.

[5] S. Arora, R. Ge, S. Sachdeva, and G. Schoenebeck. Finding overlapping communities in social networks: toward a rigorous approach. In EC, 2012.

[6] S. Asur and S. Parthasarathy. On the use of viewpoint neighborhoods for dynamic graph analysis. Technical report, Technical Report Sept 2008 OSU-CISRC-9/08-TR50, 2008.

[7] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In ACM SIGKDD, 2006.

[8] M. Balcan, C. Borgs, M. Braverman, J. T. Chayes, and S. Teng. Finding endogenously formed communities. In SODA 2013.

[9] S. Brauer and T. C. Schmidt. Are circles communities? A comparative analysis of selective sharing in google+. In ICDCS, 2014.

[10] R. Burt. Structural Holes: The Social Structure of Competition. Harvard University Press, 1995.

[11] F. Chung. A local graph partitioning algorithm using heat kernel pagerank. Internet Mathematics, 2009.

[12] M. Coscia, G. Rossetti, F. Giannotti, and D. Pedreschi. Uncovering hierarchical and overlapping communities with a local-first approach. TKDD, 2014.

[13] J. Dean and S. Ghemawat. Mapreduce: a flexible data processing tool. Commun. ACM, 2010.

[14] R. I. Dunbar. Neocortex size as a constraint on group size in primates. Journal of Human Evolution, 1992.

[15] R. I. M. Dunbar and S. G. B. Roberts. Communication in social networks: Effects of kinship, network size and emotional closeness. Personal Relationships, 2010.

[16] A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In WWW, 2015.

[17] P. Erdős and A. Rényi. On the evolution of random graphs. In Hungarian Academy of Science, 1960.

[18] M. Everett and S. P. Borgatti. Ego network betweenness. Social Networks, 2005.

[19] L. C. T. Freeman. Centered graphs and the structure of ego networks. Mathematical Social Sciences, 1982.

[20] M. Girvan and E. J. Newman. Community structure in social and biological networks. PNAS, 2002.

[21] D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In KDD, 2012.

[22] A. V. Goldberg. Finding a maximum density subgraph. Technical report, 1984.

[23] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh. WTF: the who to follow service at twitter. In WWW '13, 2013.

[24] M. Jerrum and A. Sinclair. Conductance and the rapid mixing property for markov chains: the approximation of the permanent resolved (preliminary version). In STOC, 1988.

[25] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Random graph models for the web graph. In FOCS, 2000.

[26] M. La Gala, V. Arnaboldi, M. Conti, and A. Passarella. Ego-net digger: A new way to study ego networks in online social networks. In HotSocial '12, 2012.

[27] H. Lamba and R. Narayanam. Circle based community detection. In I-CARE '13, 2013.

[28] S. Lattanzi and D. Sivakumar. Affiliation networks. In STOC 2009.

[29] J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. TKDD, 2007.

[30] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Internet Mathematics, 2009.

[31] J. Leskovec, K. J. Lang, and M. W. Mahoney. Empirical comparison of algorithms for network community detection. In WWW 2010.

[32] R. Li, C. Wang, and K. C. Chang. User profiling in an ego network: co-profiling attributes and relationships. In WWW, 2014.

[33] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. J. Am. Soc. Inf. Sci. Technol., 2007.

[34] Y. Lim and U. Kang. Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams. In KDD, 2015.

[35] J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In NIPS, 2012.

[36] T. T. Nguyen, P.-M. Hui, F. M. Harper, L. Terveen, and J. A. Konstan. Exploring the filter bubble: The effect of using recommender systems on content diversity. In WWW, 2014.

[37] B. S. Rees and K. B. Gallagher. Overlapping community detection by collective friendship group inference. In ASONAM, 2010.

[38] P. Ronhovde and Z. Nussinov. Local resolution-limit-free potts model for community detection. Phys. Rev. E, 2010.

[39] D. Schall. Link prediction in directed social networks. Social Netw. Analys. Mining, 2014.

[40] T. Schank. Algorithmic Aspects of Triangle-Based Network Analysis. PhD thesis, 2007.

[41] A. Sharma, M. Gemici, and D. Cosley. Friends, strangers, and the value of ego networks for recommendation. In ICWSM, 2013.

[42] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In WWW, 2011.

[43] G. Szekeres and H. Wilf. An inequality for the chromatic number of a graph. Journal of Combinatorial Theory, 1968.

[44] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In KDD, 2006.

[45] H. Wang and W. Li. Online egocentric models for citation networks. In IJCAI, 2013.

[46] S. Wasserman and K. Faust. Social network analysis: methods and applications. Cambridge University Press, 1994.

[47] J. Xie, B. K. Szymanski, and X. Liu. Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In ICDMW '11, 2011.

[48] J. Yang, J. J. McAuley, and J. Leskovec. Detecting cohesive and 2-mode communities indirected and undirected networks. In WSDM, 2014.

APPENDIX

A. DESCRIPTION OF CLUSTERING ALGORITHMS

Personalized PageRank partitioning (PPR). The Personalized PageRank (PPR) clustering algorithm is a well-known non-overlapping algorithm that optimizes conductance [4]. The algorithm is based on the computation of the PPR scores. For a given node v (the so-called seed), and for a probability α, the PPR score from v to u is the probability that the PPR random walk for v is in u: At each step, the PPR random for v jumps back to node v with probability $\frac{1}{2}\alpha$, and $\frac{1}{2}(1-\alpha)$, the walk moves to a neighbor of the current node uniformly at random. Finally with probability $\frac{1}{2}$ remains in u.

Our first partitioning algorithm uses PPR clustering [4] as a building block. The algorithm takes as input a target conductance β and an induced subgraph S and it works as follows. During the execution of the algorithm each node in S is labeled as *unclustered*, *clustered* or *checked*. At the beginning all the nodes in S are labeled as *unclustered*, then we iteratively pick a node v labeled *unclustered* uniformly at random and compute a cluster C_v around it using the PPR clustering on the graph induced by the nodes labeled *unclustered* or *checked*. If $\phi(C_v) \geq \beta$, we labeled all the nodes in C_v as *clustered* and we add C_v to the final clustering $\mathscr{C}$. Otherwise we labeled the node v as *checked*. We stop when all nodes are either labeled *checked* or *clustered* and we output a partitioning $\mathscr{C}$ of S composed by C plus a singleton cluster for each node labeled *checked*. In our experiments we set $\alpha = 0.15$, $\beta = 0.3$.

Personalized PageRank overlapping clustering (Over. PPR). Our second clustering algorithm is again based on the PPR random walk. The algorithm takes as input a target conductance β, an induced subgraph S, a maximum overlap threshold γ and it works as follows. For each node $v \in S$ we compute a cluster C_v around v using the PPR clustering on the graph induced by S. Then we add C_v to $\mathscr{C}$ if and only if $\phi(C_v) \geq \beta$. After this first phase, the algorithm iteratively merges together any pair of clusters in $\mathscr{C}$ whose pairwise Jaccard coefficient is more than γ percent. In our experiments we set $\alpha = 0.15$, $\beta = 0.3$ and $\gamma = 0.8$

Label Propagation Clustering via Absolute Potts Model (LP).

We used a label propagation algorithm based on the Absolute Potts Model technique [38]. The algorithm takes as input the graph to cluster $G(V, E)$ and three constant α, β and T and proceeds in as a series of rounds. Initially all nodes are assigned a unique label (the node id). At each iteration (until a convergence criteria is met) the algorithm select a random permutation of all nodes. Then nodes are evaluated in order of the permutation, one at a time.

Consider the evaluation of current node u, in a given iteration. Let $N^+(u) = x_1 \dots x_t$ be the set of the out-neighbors of u. Let $L(x_i)$ be the current label of node x_i. For a given label l, let $C_u(l)$ be the number of out-neighbors of u that have label l. Let $T(l)$ be total number of nodes in the graph with label l. The algorithm updates the label of u with the label l that maximizes the following function $f_u(l)$: $f_u(l) = C_u(l) - \alpha \cdot (T(l) - C_u(l))$ (breaking ties randomly). Intuitively the algorithm chooses the most common label in the neighborhood of u after penalizing with parameter α the number of nodes with the same label that are not connected to u.

The algorithm continues until at least one of the two following conditions is met: (i) A maximum number of iterations T is reached. (ii) In the last iteration less than a fraction β of nodes changed their label.

The algorithm is very scalable as each iteration is $O(n + m)$, n number of nodes, m number of edges in the subgraph, so the total cost is $O(k(n + m))$ for k iterations. The space is $O(n + m)$ as well. In our experiments we set $\alpha = 0.3$, $k = 20$.

Overlapping Label Propagation Clustering (SLPA). The third algorithm evaluated is a variation of the label propagation algorithm known as Speaker-Listener Propagation Algorithm SLPA [47]. This algorithm naturally generalize label propagation to generate overlapping clusters: For each node, instead of possessing a single label, the node stores all the labels received during the execution of the algorithm. This implies that at the end of the execution, a node possesses a distribution over the communities in the graph, given by the frequency with which the node has observed each label id. This distribution can be used to assign nodes to multiple communities.

More formally, the algorithm has three parameters T, α and β. At all times, each node possesses a list of possibly repeated labels that is initialized, for each node, with a unique label. For T iterations, nodes are analyzed in random order. When node u is evaluated, the node considers the list of its labels (including repetitions) and sends a label selected uniformly at random to all outgoing neighbors that consequently update their lists including the new label. Labels are never removed from the list and the list hence grows by $O(d^-(v))$ labels per iteration per each node v. At end of the procedure, if node u has received label id l more than a fraction α of the iterations, then node u joins community l. If no label has been observed for more than such fraction, than node u joins the most frequent label community.

As in the previous overlapping clustering, at the end, we merge communities whose pairwise Jaccard coefficient is larger than β. In our experiments, we set $\alpha = 5$ and $\beta = 0.8$.

Connected Components (CC). We also examine the connected components as a baseline for comparison.

Hierarchical Clustering using k-cores (K-Core). Finally, we consider the k-core-based overlapping clustering algorithm. A k-core [43] is a subset C of the nodes such that any node in C has degree $\geq k$ in the induced subgraph of C. The algorithm produces a hierarchical clustering of V by computing the k-cores of the graph in an iterative way use a different values of the threshold k.

The algorithm initializes $\mathscr{C}$ as the empty set. In the first iteration, k is equal to input parameter α, then at each iteration, k is increased by multiplying it with input parameter $\gamma > 1$.

The algorithm computes the k-core V_k of G and includes in $\mathscr{C}$ the set of all the connected components of $G[V_k]$. The method ends when all the nodes in the graph are removed for a certain k.

The algorithm takes $O(\log_\gamma(n)) = O\left(\frac{\log(n)}{\gamma-1}\right)$ iterations, which is $O(\log n)$ for constant $\gamma > 1$. The total cost of the algorithm, for constant $\gamma > 1$, is hence $O(m \log(n))$. Similarly to previous algorithm, we finally merge communities whose pairwise Jaccard coefficient is larger then β. In our experiments we set $\alpha = 1$, $\gamma = 2$ and $\beta = 0.8$.