The central community of Twitter ego-networks as a means for fake influencer detection

Nicolas Tsapatsoulis*, Vasiliki Anastasopoulou†, Klimis Ntalianis‡
*Dept. of Communication and Internet Studies, Cyprus University of Technology, Cyprus
Email: nicolas.tsapatsoulis@cut.ac.cy
†Dept. of Psychology, National and Kapodistrian University of Athens, Greece
‡Department of Marketing, University of West Attica, Greece

***Abstract*—The central community of social networks, usually represented through the highest degree k-core of the corresponding graph, is proposed here as a compact representation of large social networks. We show that the central community of egocentric social media networks, such as the ego networks on Twitter and Instagram, tell us much more about the actual influence of the ego than the whole egocentric network itself. We also propose a novel genetic algorithm for the identification of central community of egocentric social networks and we examine the importance of the proper initialisation of this algorithm. The actual Twitter ego networks we used in our experiments along with the corresponding Python code are made publicly available for anyone who wishes to use them.**

***Index Terms*—degeneracy, graph partitioning, community detection, social networks, genetic algorithms, k-core, twitter ego-networks**

I. Introduction

The k-core of a graph is a maximal subgraph in which each node has at least degree equal to k, i.e., it has at least k neighbours. The typical structure of social networks is a bushy one [1] and as a result the k-core of the corresponding graph consists of several disconnected components [2]. This is not the case, however, for egocentric networks where the k-core, for high values of k, is usually a single component organised around the ego. The k-core corresponding to the degeneracy (hereafter the *degeneracy core*), i.e., the maximum k for which a graph contains a k-core, corresponds to the densest part of graph. In many cases, and especially for large social networks like the ones created through contemporary social media such as Facebook, Instagram and Twitter, the study of the degeneracy core provides more robust measures than the study of the whole network. Furthermore, the study of the degeneracy core is one of the approaches used for the characterisation of very large networks [1].

The tremendous expansion of Online Social Networks (OSN) provides a variety of opportunities for communication, marketing and other activities among users, companies and organisations in a global scale. Social media, including Twitter and Instagram, are used on a daily basis not only for socialising in the cyberspace but also for getting information, help and recommendation. At the same time big companies and brands have a strong presence in social media in order to promote their products and services. The combination of the two led to a new form of social media marketing known as influencer marketing [3]. In OSN influencers are well connected accounts followed by thousands of other users. Influencer marketing is considered more effective than traditional marketing since influencers are more trustworthy than a business because they develop close connections with their followers. However, becoming a social media influencer is not easy and it requires a lot of effort and time. On the other hand, identification of actual influential users in OSN is also tricky. Simple centrality measures, such as the number of followers, led to the emergence of vendors selling OSN fake accounts through which people can inflate their follower counts and even their engagement. Thus, brands seeking to expand their financial circles through OSN influencers may collaborate with fake influencers. As a result, fake influencers secure sponsorships but in reality they cannot impact real users.

Fake influencers are present across all major social media platforms (Instagram, Twitter, Facebook) and the ability to identify them is critical to the success and credibility of influencer marketing through OSN. So far the proposed methods have been focusing to the detection fake accounts and little effort has been devoted to the detection of fake influencers. The current trend in fake user detection on OSN is to employ methods that utilise specific account features like date of account creation, number of posts, number of followers etc, while a fewer number of approaches, dedicated to Twitter users, involve some simple centrality measures [4].

In a previous work [5] we used empirical analysis of egocentric networks of social media users to identify fake Influencers in Twitter. For this purpose we evaluated several centrality and network characterisation measures and we showed that both of them differ significantly between legitimate and fake influencers. In any case, the use of full egocentric networks for fake influencer detection was an innovation per se. However, in practice the egocentric networks of both fake and legitimate influencers are usually very large and computing measures directly on them is not efficient. In the current study we show that central community of egocentric networks identified through the use of a genetic algorithm provides both effective and efficient means for fake influencers identification.

Genetic Algorithms (GA) are adaptive optimisation methods that resemble the evolution mechanisms of biological species [6]. They do not require the continuity of parameter space and they are able to efficiently search over a wide range

of parameters / parameter sets. In a GA, the search begins from a population of possible solutions (in our case binary strings of length equal to the number of users N, with ones denoting the users that form a cluster), and not just one possible solution.

The initial population is randomly acquired; this means that the first and major degree of diversity is introduced in this stage of the GA. However, proper initialisation of the population has been shown to tremendously speed up convergence and in most cases wins the trade-off battle with the diversity. The second and lesser degree of diversity is introduced when the mutation operator acts upon each string of the population. The whole evolution process stops after a predefined maximum number of iterations (generations) is reached.

The idea of using Genetic Algorithms for community detection in social networks is not new. It was used, basically in identifying overlapping communities or in a hierarchical manner for graph clustering in general. Here we adopt it for central community identification of follower-based social networks such as Twitter and Instagram.

II. Background and related work

While a lot of research has been reported on graph clustering (for a comprehensive review see the articles of Fortunato [7] and Malliaros & Vazirgiannis [8] for directed graphs, as well as the two recent reviews by Javed *et al.* [9] and Fortunato & Hric [10]) not much research work has been conducted for the detection of central communities of graphs and especially of graphs corresponding to egocentric networks. This trend can be attributed to three reasons: (1) the detection of central community of a graph is considered a special case of graph clustering, (2) the central community of a graph is assumed to be well approximated by its degeneracy core and the influential algorithm of Batagelj & Zaveršnik [11] for k-core detection was implemented in the majority of network analysis software including Python's *networkx* library, and (3) the usefulness of central community as an approximation of larger networks was not sufficiently pointed up.

The current work shows the importance of central community of graphs corresponding to egocentric networks for the classification of Twiitter accounts into fake and legitimate influencers. While our method for the central community identification is based on evolutionary optimisation the degeneracy core as well as k-cores corresponding to k lower than the degeneracy are used for the initialisation of the population of solutions. Both the identified central community as well as the degeneracy core give us a better approximation of the real influence of the Twitter account owners than the whole egocentric network.

Shin *et al.* [12] in their recent work investigated how the k-core structures of real-world graphs look like and which are their common patterns and anomalies. They also discuss possible exploitation of k-cores for real world applications. Their major findings are: (1) Coreness, that is the maximum k such that a vertex belongs to a k-core, is strongly correlated with node degree, (2) the degeneracy obeys a 3-to-1 power-law with respect to the count of triangles, (3) while the degeneracy cores are not cliques they do have non-trivial structures such as coreperiphery and communities. According to the authors, deviation from coreness spots anomalies in real-world graphs while the number of triangles in the graph provides a rough estimation of the degeneracy. In the current work we used the latter both as a measure to differentiate between fake and legitimate influencers as well as a lower bound for k for the k-cores that initialise our population of solutions.

In the following we present a short literature review regarding the use of Genetic Algorithms for graph clustering since the proposed central community detection approach falls within this research area. Application of natural computation inspired methods, including GAs, in community detection was recently reviewed by Zhang *et al.* [13] and the interested reader is referred there for a thorough survey. It is also important to refer to the influential algorithm of Batagelj and Zaveršnik [11] for k-core identification as this algorithm, and especially the implementation in the Python's *networkx* library, is used for population initialisation in our GA method.

In graph clustering, GAs were employed to optimise the network modularity in order to detect overlapping or even hidden communities [14]. The fact that GAs do not require the number of communities (partitions) beforehand, along with their ability to create overlapping clusters are the two most important properties that led to the widespread use of GAs for community detection in social networks.

Bui and Moon [15] were among the first researchers that applied GAs for graph partitioning. In order to improve the time to converge they used a schema preprocessing heuristic while they applied their algorithm on a variety of graph types, mainly synthetic ones, reporting comparable results with the simulated annealing method and the Kernighan - Lin [16] graph bisection algorithm in terms of efficiency. While the method by Bui and Moon is no more a state of the art one, the principles they set in their approach influenced many researchers that applied, in later years, GAs for graph partitioning.

To a great extent the variability of methods that apply GAs for graph partitioning is introduced through the optimisation criteria adopted. Pizzuti's [17] GA-NET employs the concept of *community score* to show the quality of partitioning social network graphs. The community score is the maximisation of internal links in a community structure. The efficiency of that method is enhanced through the modification of the variation operators to take into account only the actual correlations among the nodes. In this way, the research space of possible solutions is considerably reduced. As in most cases the experiments were conducted mostly on synthetic graphs.

Shang *et al.* [18] proposed a community detection method based on a genetic algorithm which takes the network's modularity Q as the objective function while prior information such as the number of community structures is also used. The optimisation method used is simulated annealing and, according to the authors, the ability of local search is improved by adjusting the parameters of the algorithm appropriately. The

method was, mainly, tested on computer-generated data while no benchmarking with other methods is reported. The fact that the number of clusters is required as a parameter for the optimisation function is also another drawback of that method.

Gurrero *et al.* [19] proposed a generational genetic algorithm, named GGA+, that includes also population initialisation while the space of solutions is searched under the guidance of network modularity as in Shang *et al.* [18]. Adaptive analysis of the characteristics of a network from different levels of detail according to analysts' needs is also supported.

Bilal and Abdelouahab [20] applied an evolutionary algorithm to repeatedly find the first community structure that maximises modularity. After that they merge the identified communities to find the final community structure that has the highest value of modularity. Identifying the first community structure that maximises modularity can be used for central community detection but there is no evidence that this community has the characteristics of the degeneracy core.

Gong *et al.* [21] initially proposed a genetic algorithm named as Meme-Net in which the optimisation criterion was the modularity density. In their algorithm, a local search climbing strategy was adopted to improve the performance of traditional GAs. In a later work they [22] used a multi-objective optimisation criterion which maximises the density of internal degrees within a cluster and minimises the density of external degrees outside the cluster, simultaneously. According to the authors, the algorithm produces a set of solutions which can represent various divisions of the network at different hierarchical levels. The number of communities is determined by the non-dominated individuals resulting from the algorithm. Experiments were contacted on both synthetic and real-world network datasets, the latter being, however, quite small.

Multi-objective criteria were also adopted by Liu *et al.* [23] for the detection of communities in signed social networks. Their approach is based on the signed similarity which takes into account the contradiction between positive and negative links. In order to account for the direct and indirect form of communities in real-world networks, a combined representation was adopted allowing the algorithm to switch between the two different representations during the evolutionary process. Owing to this representation overlapping communities can be identified. While the authors used both real-world and synthetic networks the large networks correspond to synthetic data.

Rahimi *et al.* [24] used also a multi-objective approach with the aid of particle swarm optimisation (MOPSO-Net). Kernel k-means and ratio cut were the two objective criteria that were minimised. The contribution of that work was actually the modification of particle swarm optimisation algorithm by changing the moving strategy of particles. Experiments were conducted on synthetic and real-world networks of low to medium scale. Thus, the conclusions drawn cannot be generalised to contemporary OSN.

As already explained our method emphasises on the identification of the central community of real-world large ecocentric networks mined from the Twitter. Thus, it cannot directly compared with anyone of the previously mentioned methods. We consider the central community as a compact network representation and we show that through this representation the actual influence value of the ego node to be assessed. Regarding the technicalities of the proposed GA algorithm both the objective function adopted as well as the initialisation of the population have never presented before. Finally, our experimentation is purely applied on real-world networks of large size while the corresponding Python code is provided in the Appendix.

III. Methodology

We follow the typical notation of network representation as a graph $\mathcal{G}$, i.e., $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ with $\mathcal{V}$ being the set of vertices (nodes) and $\mathcal{E}$ being the set of edges (ties between vertices).

A. Central community identification using genetic algorithms

In order to apply a genetic algorithm optimization for cluster creation we consider a set of binary vectors $\mathbf{B} = \{\vec{b}_i | i = 1, \ldots, P_N\}$, with $\vec{b}_i \in \{0,1\}^N$. Each vector $\vec{b}_i$ represents an initial cluster formation, i.e., the subgraph nodes. Thus, the length N, of vector $\vec{b}_i$ is equal to the total number of nodes in the graph while the positions of the ones in vector $\vec{b}_i$ indicate that the corresponding nodes belong to the cluster (say subgraph $\mathcal{C}_i$). According to this formulation P_N is the number of initial solutions (clusters).

In order to speed-up the rate of convergence (see a direct comparison in Figures 1 and 2) population initialisation was adopted. Binary vectors corresponding to k-cores, for various values of k, computed using the algorithm of Batagelj and Zaveršnik [11], were used along with random binary vectors of size N to create the initial population. The lowest value of k was taken as the estimated degeneracy, according to the findings of Shin *et al.* [12] (see also eq. 6).

Once the initial population has been created the process of creating new generations starts and consists, typically, of three stages:

1) A fitness value (measure of "optimality") of each string (subgraph $\mathcal{C}_i$) in the population is calculated.
2) Genetic operators, corresponding to mathematical models of simple laws of nature are applied to the population and result in the creation of a new population.
3) The new population replaces the old one.

There are three types of operations in differential evolution processing: *mutation*, *crossover* and *selection*. Mutation maintains population diversity and improves the ability of global exploration, crossover accelerates convergence speed and improves the ability of local exploration while selection has historical memory and can preserve excellent individuals.

In our case optimisation aims to identify the cluster $\mathcal{C}_{opt}$ (encoded through the binary string $\vec{b}_{opt}$) that corresponds to the central community of the graph $\mathcal{G}$ assuming that: (i) the central community has the properties of the degeneracy core, (ii) the strict definition of a k-core is relaxed so as to allow

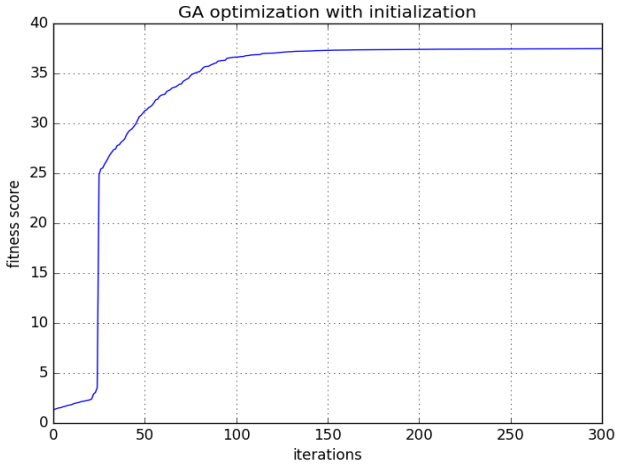

Fig. 1. Convergence rate of GA with initalization

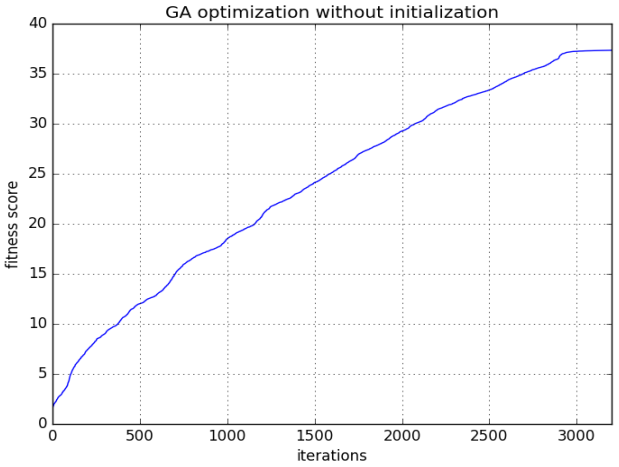

Fig. 2. Convergence rate of GA without initalization

nodes of degree lower but close to k to be included. For this purpose we define an appropriate fitness function (objective criterion) as follows:

$$F(\vec{b}_i) = F(\mathcal{C}_i) = \frac{1}{a(N_{C_i})} \sum_{v \in \mathcal{V}_{C_i}} \frac{d[v]}{N_{C_i}} \tag{1}$$

where $\mathcal{C}_i := (\mathcal{V}_{C_i}, \mathcal{E}_{C_i})$ is a subgraph of $\mathcal{G}$, N_{C_i} is the number of nodes of $\mathcal{C}_i$, $d[v]$ is the degree of node v and $a(N_{C_i})$ is a correcting factor used to account for the fact that the average degree of a graph depends on the number of its nodes. In the current work $a(N_{C_i})$ was taken as:

$$a(N_{C_i}) = \sqrt{N_{C_i}} \tag{2}$$

The realisation of the genetic operators selection, crossover and mutation is then taken as follows:

Selection for reproduction. The fitness function $F(\vec{b}_i)$ is used in the classical "roulette" wheel reproduction operator that gives higher probability of reproduction to the strings with better fitness according to the following procedure:

1) An order number, q, is assigned to the population strings. That is q ranges from 1 to P_N, where P_N is the size of population.
2) The sum of fitness values (F_{sum}) of all strings in the population is calculated.
3) The interval $[0\ F_{sum}]$ is divided into P_N sub-intervals each of one being $[SF_{q-1}\ SF_q]$
where $SF_{q-1} = \sum_{j=1}^{q-1} F(\vec{g}_i)$ for $q > 1$ and $SF_{q-1} = 0$ for $q = 0$ or $q = 1$,
and $SF_q = \sum_{j=1}^{q} F(\vec{g}_i)$ for every q.
4) A random real number R_0 lying in the interval $[0, F_{sum}]$ is selected.
5) The string having the same order number as the subinterval of R_0 is selected
6) Steps (iv) and (v) are repeated P_N times in order to produce the intermediate population to which the other genetic operators will be applied.

Crossover. Given two strings $\vec{b}_i$ and $\vec{b}_j$ (parents) of length N an integer number r is randomly selected. The two strings retain their gene values up to gene r and interchange the values of the remaining genes creating two new strings (offsprings).

Mutation. This operator is applied to each gene of a string and it alters its content, with a small probability. The mutation operator is actually a random number that is selected and depending on whether it exceeds a predefined limit it changes the value of a gene.

B. Evaluation Metrics for the detection of fake influencers

Once the central community of a graph, corresponding either to a fake or to a legitimate Twitter influencer, the following evaluation metrics are computed. We avoided using measures that have been already discussed in our previous works [5] because the majority of them full egocentric network to be known.

1) Ego's in degree: The *in degree* of the *ego* in celebrity and fake influencer networks is usually large compared to the *out degree* of the *ego* or the number of vertices in the network. We include this measure here, denoted as $\check{d}[v_e]$, in order to verify the previous assumption. By v_e we denote the ego.

2) Average degree: The average degree, given by eq. 4, is a measure for global network characterisation. Typically fake users have low in degree and as a result one expects that fake influencer egocentric networks will have a rather low average degree.

$$\bar{d} = \frac{\sum_{v \in \mathcal{V}} d[v]}{N} \tag{3}$$

where $\mathcal{V}$ is the set of network vertices, $d[v]$ indicates the degree of vertex v, and N is the number of nodes of graph $\mathcal{G}$.

The average degree of the central community, denoted as $\bar{d}_C$, is computed in a similar manner:

$$\bar{d}_C = \frac{\sum_{v \in \mathcal{V}_C} d[v]}{N_C} \tag{4}$$

3) Relative average degree of central community: With the aid of average degree of the central community we define a similar normalised measure that accounts for the length of the full network in terms of the number of nodes. The relative average degree of central community is given by eq. 5 below.

$$r_{\bar{d}} = \frac{log_2 \bar{d}_C}{log_2 N} \tag{5}$$

4) Estimated degeneracy: According to the work of Shin *et al.* [12] the degeneracy, say k_{max}, of a real-world network can be estimated by the total number T_N of triangles in the graph via a power law. It appears that in egocentric networks this estimation is not so accurate, however, the highest the deviation is the more likely the ego is a real influencer.

In the current work we estimate the degeneracy with the aid of formula 6. This estimation provides a lower bound for the actual degeneracy number for legitimate influencers and other typical egocentric network but this is not the case for fake influencers. Nevertheless, the estimated degeneracy is used for the initialisation of the pool of solutions for the genetic algorithm as already explained in Section III-A.

$$\hat{k}_{max} = 3 \cdot log_2 T_N \tag{6}$$

Two additional estimations of the degeneracy are also proposed in the current work (see also the discussion in Section V). They are based on the average degree of the whole network, $\bar{d}$, and the central community, $\bar{d}_C$:

$$\hat{k}^F_{max} = 1.15 \cdot \bar{d} \tag{7}$$

$$\hat{k}^C_{max} = 0.71 \cdot \bar{d}_C \tag{8}$$

5) Relative degeneracy estimators difference: As discussed in Section III-A the relative difference, shown in eq. 9, among the degeneracy estimators of eq. 8 and 7 varies significantly among the fake and legitimate influencer groups.

$$r_k = \frac{|\hat{k}^C_{max} - \hat{k}^F_{max}|}{\hat{k}^C_{max}} \tag{9}$$

IV. Dataset and data collection

The dataset of our experiments consists of egocentric networks directly mined from the Twitter with the aid of Twitter API[1]. We started first by identifying Twitter accounts that correspond to fake influencers by following the approach of Zenonos *et al.* [5]: A Twitter account (*@andreast88*) was registered and 1000 fake followers were bought from three different vendors. Candidate fake influencers were identified by checking other Twitter accounts that the fake followers of *@andreast88* also follow. Note that due to the recent measures taken by Twitter regarding fake accounts most of *@andreast88* eventually disappeared. However, we had already identified thousands of candidate fake influencers among which 45 were manually selected and crawled during the period March 2018 - October 2018. As in [5] we adopted the following working definition for fake influencer: "Fake influencer in Twitter is an account whose a great proportion (higher than 50%) of followers are fake".

Legitimate influencers were selected among politicians, journalists, TV personas, football players and marketing specialists from a variety of countries including Cyprus, Turkey, Italy, UK and USA. For the purpose of the current experiment we focused on accounts whose number of followers match that of fake influencers so as to have a fair comparison (see Table I). We should mention here that a Twitter egocentric network consists of the ego (the Twitter account under investigation), its alters (her/his friends and followers) and all ego-alter and alter-alter ties. Thus, mining the egocentric network corresponding to a Twitter account is a time consuming procedure. This is the reason we believe that even the dataset its own is an important contribution to the research dealing with graph partitioning of real-world social networks. Thus, the final dataset we used in our experiments, consisting of 20 Twitter egocentric networks corresponding to 11 legitimate and 9 fake influencers, was transformed into Pajek[2] format and is publicly available through our website at https://irci.eu/.

V. Results and discussion

The values of all measures described in Section III-B, along with some basic characteristics of the full graphs corresponding to all 20 egocentric networks that were investigated, are presented in Table I. A two independent means two tail t-test was conducted, on all measures presented in Table I, to identify statistical significance and the corresponding p-values among the two groups, i.e., legitimate and fake influencers. We observe, first, that there is no significant difference on the size of the compared networks, thus, the comparison on the rest of the measures is fair. It also appears that there is no significant difference on the number of ego's followers. In both legitimate and fake influencers their followers consist the great majority of network's nodes.

All the measures computed from the full graph present statistical significance among fake and legitimate influencer groups. However, we especially note the huge deviation between the actual degeneracy, k_{max}, and the estimated one, $\bar{k}_{max}$, in the case of the legitimate influencers. Also, we observe that the degeneracy is, in all but one cases, overestimated in the fake influencers while the situation is reversed in the legitimate influencers. Thus, the ratio $k_{max}/\bar{k}_{max}$ is good indication of the influencing potential of a Twitter account. The problem, however, is the time complexity that is required for the estimation of degeneracy and especially for the computation of the number of triangles T_N (see also eq. 6) in a large network. On the other hand, we observe a strong correlation, i.e., $\rho(\bar{d}, k_{max}) = 0.946$, between the degeneracy and average degree in the legitimate influencers. This means that a rough estimation, $k_{max} \approx 1.15 \cdot \bar{d}$, of the degeneracy can be obtained from the average degree which is a very simple to compute

[1] https://developer.twitter.com/en/docs.html

[2] http://mrvar.fdv.uni-lj.si/pajek/

TABLE I
EVALUATION METRICS

	Full Graph								Central Community				Combined		
Graph	N	$\check{d}[v_e]$	k_{max}	$\hat{k}_{max}$	$\bar{d}$	$k_{max}/\hat{k}_{max}$	$\hat{k}^F_{max}$	$I_R(x10^3)$	N_C	$\bar{d}_C$	$\hat{k}^C_{max}$	$I^C_R(x10^3)$	$r_{\bar{d}}$	r_k	$\hat{k}_{max}/\bar{d}_C$
F01	644	409	47	50	21.08	0.94	25	4.3	59	65.66	46	1.9	0.6470	0.4894	0.7615
F02	924	863	45	51	17.25	0.88	21	7.4	59	60.31	42	1.8	0.6003	0.5349	0.8456
F03	1066	1032	31	54	29.84	0.57	36	15.4	125	50.94	36	3.2	0.5638	0.0556	1.0601
F04	2409	2002	44	50	7.95	0.88	10	8.0	57	62.84	44	1.8	0.5317	0.8000	0.7957
F05	3058	2934	24	43	3.96	0.56	5	5.9	27	33.78	24	0.5	0.4386	0.7917	1.2729
F06	4177	3992	52	50	4.30	1.04	5	8.6	A61	70.52	49	‘2.2	0.5105	0.9000	0.7090
F07	4696	3358	39	53	11.40	0.74	14	19.1	331	56.41	39	9.3	0.4770	0.6750	0.9395
F08	4857	4748	21	41	2.58	0.51	3	6.1	29	27.45	19	0.4	0.3902	0.8421	1.4936
F09	4862	4682	25	41	2.65	0.61	3	6.2	27	34.07	24	0.5	0.4156	0.8750	1.2034
L01	585	520	63	56	55.71	1.13	67	14.5	79	88.73	62	3.5	0.7040	-0.0635	0.6311
L02	975	823	147	66	125.12	2.23	150	51.5	187	212.47	149	19.9	0.7786	0.0066	0.3106
L03	1059	1038	145	66	125.68	2.20	151	65.2	256	225.83	158	28.9	0.7781	0.0563	0.2923
L04	1310	1112	192	71	217.33	2.70	261	120.8	416	310.20	217	64.5	0.7993	-0.1864	0.2289
L05	1660	1367	146	68	133.42	2.15	160	91.2	193	201.17	141	19.4	0.7154	-0.1189	0.3380
L06	1919	1873	326	71	308.27	4.59	370	288.7	507	434.04	304	110.0	0.8034	-0.2013	0.1636
L07	2035	1759	255	73	212.77	3.49	255	187.1	291	343.00	240	49.9	0.7663	-0.0451	0.2128
L08	2177	1928	190	73	162.74	2.60	195	156.9	365	302.86	212	55.3	0.7434	0.0930	0.2410
L09	2226	1981	326	77	299.74	4.23	360	296.9	379	444.02	311	84.1	0.7909	-0.1429	0.1734
L10	3942	3807	300	77	222.64	3.90	267	423.8	378	412.14	288	77.9	0.7273	0.0887	0.1868
L11	4096	4052	332	78	245.84	4.26	295	498.1	386	440.82	309	85.1	0.7320	0.0575	0.1769
p-value	.1554	.2043	.00001	<.00001	<.00001	<.00001	n/a	.0021	.0003	<.00001	n/a	.0002	<.00001	<.00001	<.00001
signif.	No	No	****	****	****	****		**	***	****		***	****	****	****

measure. At the same time, as expected, the degeneracy is strongly correlated, i.e., $\rho(\bar{d}_C, k_{max}) = 0.9949$, with the average degree, $\bar{d}_C$, of the central community for every graph and can be approximated as $k_{max} \approx 0.71 \cdot \bar{d}_C$. Thus, the deviation between those two estimations of the degeneracy can be used for the identification of fake influencers as it can be easily seen in the r_k column of Table I.

The average degree of the central community $\bar{d}_C$ is also a reliable metric for differentiating fake and legitimate influencers. We observe that, despite the differences in the number of nodes of the corresponding full graphs, even the largest average degree of central communities of the fake influencer graphs is well below the smallest average degree of central communities of the legitimate influencers. This observation is further verified by observing the values of the relative average degree $r_{\bar{d}}$ in Table I. Regarding the latter it is interesting to note the stability of that measure in the legitimate influencers group.

A. Influence range

Our major claim in the current work is that through the central community of a Twiiter egocentric network we can reliably measure the actual influence of the ego. As already explained simple centrality measures do not work well for the estimation of the influence of a Twitter account due to the presence of fake or inactive followers. Thus, any measure of influence should consider not only the ego's followers but the followers of the followers as well. A rough estimation of this number is obtained by multiplying ego's number of followers, denoted $\check{d}[v_e]$ herein, with the half of the average degree $\bar{d}$ as in eq. 10.

$$I_R = 0.5 \cdot \check{d}[v_e] \cdot \bar{d} \tag{10}$$

We argue that a more accurate estimation of the influence range can be obtained through the central community of the graph according to eq. 11. The correlation, $\rho(I_R, I^C_R)$, in the legitimate influencer and fake influencer groups, $\rho = 0.8139$ and $\rho = 0.8820$ respectively, supports our claim.

$$I^C_R = 0.5 \cdot N_C \cdot \bar{d}_C \tag{11}$$

VI. CONCLUSION & FURTHER WORK

In this paper we ave experimentally shown that the central community of egocentric Twitter networks can provide reliable measures regarding ego's influential value. Some innovative measures, based on estimations of the degeneracy, were proposed for effective identification of fake influencers. In addition, a new metric called influence range was proposed to represent the actual influential value of the ego based on characteristics of the central community. The central community is identified through a genetic algorithm which is described in detail in the current paper. Both the fitness function used as well as the initialisation of the population are introduced here for the first time.

A side, but not negligible, outcome of this study is the creation of a new dataset of egocentric networks of fake and legitimate influencer Twitter accounts. This dataset is publicly available to the research community through our website.

Our near future plans are focusing on the use of triadic census characteristics of the ego graphs to identify differences between fake and legitimate influencers. Our emphasis will

be given to triads encoded as 201 in the MAN (Mutual Asymmetric Null) labelling scheme. Triads of this kind reflect ego's brokerage value and we anticipate that will have impact on ego's influential value as well.

ACKNOWLEDGMENT

The authors acknowledge research funding from the European Unions Horizon 2020 research and innovation programme under the Marie Skodowska-Curie ENCASE project, Grant Agreement No. 691025.

REFERENCES

[1] S. P. Borgatti and D. S. Halgin, "On network theory," *Organization Science*, vol. 22, no. 5, pp. 1168–1181, Sep. 2011.

[2] C. Giatsidis, F. D. Malliaros, D. M. Thilikos, and M. Vazirgiannis, "Corecluster: A degeneracy based graph clustering framework," in *Proceedings of the 28th conference of the Association for the Advancement of Artificial Intelligence*, ser. AAAI. AAAI Press, 2014, pp. 44–50.

[3] S. Lei, S. Maniu, L. Mo, R. Cheng, and P. Senellart, "Online influence maximization," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15. ACM, 2015, pp. 645–654.

[4] A. Mehrotra, M. Sarreddy, and S. Singh, "Detection of fake twitter followers using graph centrality measures," in *Proceedings of the 2nd International Conference on Contemporary Computing and Informatics*, Dec 2016, pp. 499–504.

[5] S. Zenonos, A. Tsirtsis, and N. Tsapatsoulis, "Twitter influencers or cheated buyers?" in *Proceedings of the 3rd IEEE Cyber Science and Technology Congress*, ser. CyberSciTech 2018. IEEE, 2018, pp. 236–242.

[6] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[7] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75 – 174, 2010.

[8] F. D. Malliaros and M. Vazirgiannis, "Clustering and community detection in directed networks: A survey," *Physics Reports*, vol. 533, pp. 95 – 142, 2013.

[9] M. A. Javed, M. S. Younis, S. Latif, J. Qadir, and A. Baig, "Community detection in networks: A multidisciplinary review," *Journal of Network and Computer Applications*, vol. 108, pp. 87 – 111, 2018.

[10] S. Fortunato and D. Hric, "Community detection in networks: A user guide," *Physics Reports*, vol. 659, pp. 1 – 44, 2016.

[11] V. Batagelj and M. Zaveršnik, "Fast algorithms for determining (generalized) core groups in social networks," *Advances in Data Analysis and Classification*, vol. 5, no. 2, pp. 129–145, jul 2011.

[12] K. Shin, T. Eliassi-Rad, and C. Faloutsos, "Patterns and anomalies in k-cores of real-world graphs with applications," *Knowledge and Information Systems*, vol. 54, no. 3, pp. 677–710, Mar. 2018.

[13] W. Zhang, R. Zhang, R. Shang, J. Li, and L. Jiao, "Application of natural computation inspired method in community detection," *Physica A: Statistical Mechanics and its Applications*, vol. 515, pp. 130 – 150, 2019.

[14] K. He, Y. i. Li, S. Soundarajan, and J. E. Hopcroft, "Hidden community detection in social networks," *Information Sciences*, vol. 425, pp. 92 – 106, 2018.

[15] T. N. Bui and B. R. Moon, "Genetic algorithm and graph partitioning," *IEEE Transactions on Computers*, vol. 45, no. 7, pp. 841–855, jul 1996.

[16] V. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Systems Technical Journal*, vol. 49, pp. 291–307, feb 1970.

[17] C. Pizzuti, "Ga-net: A genetic algorithm for community detection in social networks," in *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature: PPSN X*. Springer-Verlag, 2008, pp. 1081–1090.

[18] R. Shang, J. Bai, L. Jiao, and C. Jin, "Community detection based on modularity and an improved genetic algorithm," *Physica A: Statistical Mechanics and its Applications*, vol. 392, no. 5, pp. 1215–1231, 2013.

[19] M. Guerrero, F. G. Montoya, R. Baos, A. Alcayde, and C. Gil, "Adaptive community detection in complex networks using genetic algorithms," *Neurocomputing*, vol. 266, no. C, pp. 101–113, nov 2017.

[20] S. Bilal and M. Abdelouahab, "Evolutionary algorithm and modularity for detecting communities in networks," *Physica A: Statistical Mechanics and its Applications*, vol. 473, pp. 89 – 96, 2017.

[21] M. Gong, B. Fu, L. Jiao, and H. Du, "Memetic algorithm for community detection in networks," *Physical Review E*, vol. 84, p. 056101, Nov 2011.

[22] M. Gong, L. Ma, Q. Zhang, and L. Jiao, "Community detection in networks by using multiobjective evolutionary algorithm with decomposition," *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 15, pp. 4050 – 4060, 2012.

[23] C. Liu, J. Liu, and Z. Jiang, "A multiobjective evolutionary algorithm based on similarity for community detection from signed social networks," *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2274–2287, 2014.

[24] S. Rahimi, A. Abdollahpouri, and P. Moradi, "A multi-objective particle swarm optimization algorithm for community detection in complex networks," *Swarm and Evolutionary Computation*, vol. 39, pp. 297 – 309, 2018.

APPENDIX

Here we provide the full Python code that allows anyone who wishes to re-run the experiments and test their validity. The graphs as Pajek (see http://vlado.fmf.uni-lj.si/pub/networks/pajek/) files are also publicly available through https://irci.eu/.

1) Example of code execution commands:

```
>>> exec(open('GA_coreSubgraph.py').read())
>>> import networkx as nx
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> G = nx.read_pajek('intial_graphs/F01.net')
>>> G = nx.DiGraph(G)
>>> G.remove_edges_from(G.selfloop_edges())
>>> [new_pop, min_scores] = GA_optimization(G,
    0, np.zeros([250,G.number_of_nodes()]), 5)
>>> [new_pop, min_scores] = GA_optimization(G,
    1, new_pop, 0.05)
>>> scores = [SubGraph_ADscore(G,new_pop[i,:])
    for i in np.arange(new_pop.shape[0])]
>>> k = np.argmax(scores)
>>> C1 = [G.nodes()[i] for i in
    np.arange(G.number_of_nodes()) if
    new_pop[k,i]==1]
>>> Q1 = G.subgraph(C1)
>>> C = list(nx.k_core(G, k=192,
    core_number=None))
>>> Q = G.subgraph(C)
>>> G.number_of_nodes()
>>> np.average(list(Q.degree().values()))
>>> nx.density(Q)
>>> centralization(G)
>>> nx.write_pajek(Q1,'cores/L04_core412.net')
```

2) The python functions used are given below (contents of file *GA_coreSubgraph.py*)

```
import networkx as nx
import numpy as np
import bisect

def GA_optimization(G, init, population, T):
  N = G.number_of_nodes()
  d =
      int(np.average(list(G.degree().values())))
  number_of_solutions = 2*d
  # percentage of subgraph nodes
  b = round(0.4/(np.log(N))*N)
  pop_dims = (number_of_solutions,N)
  # Create the initial population
  if init>0:
    new_population = int_initialization(G,
        population)
  else:
    init_population = np.random.choice([0, 1],
        size=pop_dims, p=[1-b/N, b/N])
    new_population = init_population
  # find the scores of the population
  scores =
      [SubGraph_ADscore(G,new_population[i,:])
      for i in
      np.arange(new_population.shape[0])]
  # create the wheel
  wheel = np.cumsum(scores)
  wheel = wheel/max(wheel)
  min_scores=[]
  while (max(scores)-min(scores))>T:
    [px, py] = mate_pair(wheel)
    [x,y] = offsprings(new_population, px, py)
    new_population = generation(G,
        new_population, x, scores)
    new_population = generation(G,
        new_population, y, scores)
    [scores, wheel] = update(G, new_population)
    print('max_score: ', max(scores))
    print('min_score: ', min(scores))
    min_scores += [min(scores)]
  return new_population, min_scores

def int_initialization(G, new_population):
  N = G.number_of_nodes()
  d =
      int(np.average(list(G.degree().values())))
  m = d
  C1 = list(nx.k_core(G, k=m,
      core_number=None))
  lc = len(C1)
  k=0
  while lc>m:
    index_core = [i for i in np.arange(N) if
        G.nodes()[i] in C1]
    opt_vector = [0+int(i in index_core) for i
        in np.arange(N)]
    if (m//2==m/2):
      new_population[k,:]=opt_vector
    else:
      new_population[k,:]=mutation(opt_vector)
    m+=1
    k+=1
    print(k, m)
    C1 = list(nx.k_core(G, k=m,
        core_number=None))
    lc = len(C1)
  new_population = new_population[:m-d+10,:]
  return new_population

def SubGraph_ADscore(G,b_vector):
  Nc = sum(b_vector)
  N = len(b_vector)
  subset = [G.nodes()[i] for i in np.arange(N)
      if b_vector[i]>0]
  C=G.subgraph(subset)
  return np.average(list(C.degree().values()))
      / np.sqrt(Nc)

def mate_pair(wheel):
# select the parents
  x = np.random.uniform(low=0,
      high=1,size=(1,1))[0][0]
  y = np.random.uniform(low=0,
      high=1,size=(1,1))[0][0]
  px = bisect.bisect(wheel, x)
  py = bisect.bisect(wheel, y)
  while px==py:
    y = np.random.uniform(low=0,
        high=1,size=(1,1))[0][0]
    py = bisect.bisect(wheel, y)
  return px, py

def best_mate_pair(wheel, scores):
  px = np.argmax(scores)
  py = np.argmax(scores)
  while px==py:
    y = np.random.uniform(low=0,
        high=1,size=(1,1))[0][0]
    py = bisect.bisect(wheel, y)
  return px, py

def offsprings(population, px, py):
  N = population.shape[1]
  crosspoint =
      int(round(np.random.uniform(low=0,
      high=N-1, size=(1,1))[0][0]))
  off1 = list(population[px,:crosspoint]) +
      list(population[py,crosspoint:])
  off2 = list(population[py,:crosspoint]) +
      list(population[px,crosspoint:])
  return mutation(off1), mutation(off2)

def mutation(b_vector):
  N = len(b_vector)
  genes_mutated = N//3000+1
  genes = np.random.uniform(low=0, high=N-1,
      size=(1,genes_mutated))[0]
  genes = [int(round(gene)) for gene in genes]
  for gene in genes:
    prob = np.random.uniform(low=0, high=1,
        size=(1,1))[0][0]
    b_vector[gene] =
        int(round(abs(b_vector[gene]-prob)))
  return b_vector

def generation(G, population, x, scores):
  x_score = SubGraph_ADscore(G,x)
  min_index = np.argmin(scores)
  if x_score>scores[min_index]:
    population[min_index,:]=x
    scores[min_index]=x_score
  return population

def update(G, population):
  scor = [SubGraph_ADscore(G,population[i,:])
      for i in np.arange(population.shape[0])]
  wheel = np.cumsum(scores)
  wheel = wheel/max(wheel)
  return scor, wheel

def centralization(G):
  d_max = max(G.in_degree().values())
  g=G.number_of_nodes()
  e=G.number_of_edges()
  return (g*d_max-e)/(e*(g-1))
```