Node Centrality Awareness via Swarming Effects

Decebal Constantin Mocanu*, Georgios Exarchakos* and Antonio Liotta*
*Department of Electrical Engineering
Eindhoven University of Technology, The Netherlands
Email:{d.c.mocanu, g.exarchakos, a.liotta}@tue.nl

***Abstract*—Centralization is a weakness in large scale dynamic topologies and, thus, collaboratively electing at runtime the most impactful (central) nodes is necessary to ensure reliability. However, little has been achieved in measuring the centrality of nodes in an accurate, fast, decentralized and with low overhead method. This paper proposes a swarm-inspired approach (DANIS) to detect the nodes that would most impact the network connectivity if removed. The idea lies on the trivial fact that the more accessible a node is, the more resources per time unit it loses. Experiments on random, scale-free and small-world graph topologies indicate that DANIS achieves higher accuracy, faster convergence and fewer communication overhead compared to other methods.**

I. Introduction

A plethora of emerging applications on low-power lossy wireless sensor and actuator networks require reliable self-monitoring capabilities [1]. The normal operation competes with the monitoring system for the same (network, computation, energy) resources. Building communication paths over nodes with few resources may e.g. reduce the packet delivery rate, increase the transmission power or number of retransmissions, hence, speeding up the battery depletion. On the contrary, allocating enough resources to the nodes that maximize the robustness and effectiveness of the monitoring system may also release resources for the normal operation.

This paper reflects an ongoing work towards a decentralized adaptive method for detecting those nodes the failure of which would have the highest impact (centrality) to the reliability of the monitoring system. Spontaneous and fast changes (nodes rewire, join and leave frequently) set extra requirements for the node election method. The robustness of the monitoring system increases with the accuracy of picking as quickly as possible the most "critical" nodes.

Though a number of techniques have been proposed for detecting the node centrality, to the best of our knowledge, none has focused on all of decentralization, accuracy, speed and resourcing. The proposed method, namely *Decentralized Assessment of Node's Importance in networks using Swarm intelligence* (DANIS), is a decentralized algorithm relying on nomadic, reusable and non-replicable resources discovered and fetched by ants starting at various nodes. The evolution of resource concentration in each node over time gives a more accurate estimate of the centrality of nodes within the network. The fundamental idea behind is that nodes of high centrality tend to lose their resources faster.

Swarm intelligence, besides its inherently decentralized features, exhibits adaptivity capabilities fitting in large scale dynamic environments. The problem has been modeled as a multi-nest, food multi-source competitive environment. That is, each nest (node) triggers own ants to forage and fetch/steal food from other nests. The experimentation analyzed here lies on static graphs and our contributions focus on proving the concept and stressing the potential of the approach. Namely, compared to other methods [2], [3], [4], [5], DANIS can achieve higher accuracy in detecting the nodes of high centrality, faster convergence and fewer communication messages.

The paper is organized as follows. The background and related work section provides an overview of the complex networks and swarming concepts used by DANIS as well as related approaches with similar targets. DANIS mechanism is analyzed in section three and the experiments setup and results are illustrated in section four. The paper concludes with a discussion and future work.

II. Background and Related Work

In this section, we introduce the basic mathematical notations and types of Complex Networks (CN). We present some widely used centralized and decentralized methods to measure the centrality in CN, and the basic principles of Swarm Intelligence, in the benefit of the non-specialist reader. A comprehensive discussion about Complex Network can be found in [6].

A. Complex Networks

Complex networks are graphs characterized by non-trivial features. Formally, any arbitrary network it is an object which contains nodes (or vertexes) and directed or undirected edges between nodes. For simplicity, let us introduce the following standard notations:

- n represents the total number of nodes in the graph.
- m represents the total number of edges in the graph.
- $\mathbf{V} = \{v_1, v_2, ..., v_n\}$ a set with all nodes.
- $E = (v, w)$ is an undirected edge between the nodes v and w[1], where $v \in \mathbf{V}$ and $w \in \mathbf{V}$.
- $\mathbf{E} = \{E_1, E_2, ..., E_m\}$ a set with all edges.
- $\mathscr{G} = (V, E)$ a graph which has $\mathbf{V}$ nodes and $\mathbf{E}$ edges.
- $\mathbf{A}$ represents the adjacency matrix of dimensions $\{n \times n\}$ for $\mathscr{G}$, where each element $A_{ij} = 1$ if the edge (i, j) exist, and $A_{ij} = 0$ otherwise.
- $d(v)$ represents the total number of edges adjacent to node v and it is known as the degree of node v, where $v \in \mathbf{V}$.
- $\Gamma_v = \{w_1, ..., w_\gamma\}$ is the neighborhood of node v, where $v \in \mathbf{V}$, $w_j \in \mathbf{V}$, and the edge (v, w_j) exist.
- $s(v, w)$ represents the amount of hops on the shortest path between nodes v and w, where $v \in \mathbf{V}$ and $w \in \mathbf{V}$.

[1]Please note that in undirected graphs $E = (v, w) = (w, v)$

- $P(k) = n_k/n$ known as the degree distribution, is the probability distribution of the nodes degree over the whole network. More exactly, it is defined as the fraction of nodes with degree k over the total number of nodes.

Mainly, based on their properties, there are three classes of networks studied in the literature, as it is presented further.

1) Erdös-Rényi Random Graph [7] ($\mathscr{G}_{ER}$): It consists in n nodes, where each pair of nodes can be connected by an edge with a probability $p \in [0, 1]$, independently on the other edges. By using the aforementioned property, and creating the $\mathscr{G}_{ER}$ dynamically, one may obtain a graph which has no particular structures. Due to the assumption that each edge is independent, might be inappropriate to model real-world phenomenas with $\mathscr{G}_{ER}$. Hence, the "Barabási-Albert" and "Watts-Strogatz" models, discussed next, are widely used in real networks modeling.

2) Scale-Free networks ($\mathscr{G}_{SF}$): The degree distribution is a power law. Barabási et al. in [8] mapped a part of the World Wide Web topology in such a network. Also, they proposed an algorithm that uses preferential attachment to build this type of graphs. In short, this means that the nodes with high degree, called "hubs" in the literature, are favored to obtain new connections when a new node is added to the graph.

3) Small-World networks ($\mathscr{G}_{SW}$): are graphs in which each node, can be reached from any other node in a small number of steps. In other words, the shortest path between any two nodes has a length of $\propto log(n)$.

B. Centrality in Complex Networks

Centrality is a measure to assess how important are individual nodes in a network and how they can affect their neighborhood or even the whole network. However, there is no clear way to define the "centrality" in graphs. In the literature, there are more ways to calculate node's centrality, each of them being able to provide different metrics, but mainly they can be split in two main approaches: i) centralized methods and ii) decentralized ones.

1) Betweenness centrality: and its variants are ones of the most discussed ways to measure the node's importance in the literature [2]. It quantifies how a vertex lies on the path between other vertexes. Formally, for a node $v \in \mathbf{V}$ this can be written as:

$$C_{be}(v) = \sum_{w,u \in \mathbf{V}} \frac{\sigma_{w,u}(v)}{\sigma_{w,u}} \tag{1}$$

where $\sigma_{w,u}(v)$ represents the number of shortest paths from node w to node u which are passing through the node v, and $\sigma_{w,u}$ represents the total amount of shortest path from w to u. The computational complexity of the original algorithm is $\mathcal{O}(n^3)$, making it unsuitable for large networks. For this reason, in the last period, some approximations of betweenness centrality were proposed (e.g. [9]).

2) Second order centrality: is a novel form of node's centrality, calculated in a decentralized way, and proposed by Kermarrec et al. in [3]. The algorithm is based on a random walk in the graph, which starts from a random chosen node, and is running continuously. After the random walk is visiting each node for at least three times, in each vertex the standard deviation of the steps needed by the random walk to return to it is computed. In their paper, the authors showed that the computed standard deviation for a node reflects its relative importance in the network.

3) Closeness centrality: considers important the nodes which are closer, as much as possible, to the other nodes in the network. As a consequence, if some information will be spread in the network from the nodes which have a high closeness centrality, it will cover faster the whole graph. In this paper, we will note the closeness centrality for any arbitrary node $v \in V$ with $C_{cl}(v)$. It can be calculated using the formula:

$$C_{cl}(v) = \Big(\sum_{w \in \mathbf{V}} s(v, w) \Big)^{-1} \tag{2}$$

4) DACCER: is a decentralized algorithm to measure the centrality of nodes in networks, proposed by Wehmuth and Ziviani in [4]. The main idea is that each node is computing his centrality based on the information acquired from his vicinity. The authors showed that a two-hop vicinity reflects well the closeness centrality.

5) Eigenvector centrality [5]: considers important the nodes which have important neighbors. A variant of it, is used by Google PageRank algorithm. For any node $v \in \mathbf{V}$, it can be calculated with:

$$C_{ei}(v) = \lambda_1^{-1} \sum_{w} A_{vw} C_{ei}(w) \tag{3}$$

where λ_1 is the largest eigenvalue of $\mathbf{A}$.

C. Swarm Intelligence

Though the centrality of a node is a local value, its calculation needs to take into account the complete topology. DANIS models the election of high centrality nodes as a distributed exploration problem. Every node needs to find those which lose their resources faster than any other. Swarm intelligence (SI) is a suitable approach for solving it due to its simplicity, decentralization and adaptiveness features.

There have been significant efforts to mimic the foraging behavior of ants in engineered networks for traffic routing problems [10], [11]. However, routing is a point-to-point or point-to-multipoint path optimization problem whereas centrality assessment is a multipoint-to-multipoint problem. Limited effort has been put to model the multi-nest environments [12]. As stated in [12], deploying exploration processes from multiple nests reduces the probability of path stagnation and allows for more adaptiveness. A well-known class of swarm algorithms is based on the Ant Colony Optimization (ACO) [13] which reinforces a successful path by increasing the trails on it and decays the trails over time to allow for adaptiveness.

Trail decay mechanisms slow down the unlearning process in case of fast changes in the network. Based on this observation, DIP [14] and Stalkers [15] have proposed decentralized foraging algorithms relying on the previous explorations. Stalkers have used nomadic reusable non-replicable resources which migrate from node to node when released and requested. It is this very principle that DANIS reuses: resources can be either occupied or free and migrate from node to node upon request from deployed ants.

III. DANIS

In this section, we discuss the intuition and formulation of DANIS (*Decentralized Assessment of Node's Importance in networks using Swarm intelligence*), and we give its mathematical model.

A. Intuition & Formulation

Based on the intuition that in large scale networks the importance of nodes can not be assessed accurately using centralized algorithms, we assume that the collaborative behavior of decentralized and self-organized swarms will reflect better the node's centrality. Naturally, the network's dynamics can be mapped into a swarm exploration problem, as it is shown further. Inspired by Ant Colony Optimization [13], we consider a two levels swarm. The top level consists of nests, each vertex of the network being practically a nest which contains nomadic, reusable and non-replicable virtual resources, namely *vres*. On the lower level, each nest has her own ants. An ant is a basic agent defined by two states: (i) *empty*, and (ii) *loaded*. Also, has three functionalities (i) it can move within the network to discover *vres*, (ii) can harvest *vres*, and (iii) bring the harvested *vres* to its home nest. After *vres* is stored in the nest by an ant, it becomes immediately available for the other ants to harvest it. In comparison to the classical swarm approaches, the ants are not communicating between them using pheromones. The nests and the ants perform just local interactions, and the global interactions within the whole system are ensured by the migration of *vres*. Hence, the ants walk randomly the graph, to stay adaptive and pick any variations to the resource distribution across the graph.

Our intuition is that the nodes with high centrality will loose their resources faster than nodes with low centrality which will accumulate resources in time. This can be seen, to some extent, as the inverse of the information spreading problem in CN. This evolution of resource concentration reflects accurately the node centrality.

B. Algorithm

In the initialization phase of DANIS, each node is associated with a nest. To each nest, is allocated the same amount of $n_{\mathcal{V}}$ *vres*, and the same number of ants n_a, each ant being in the *empty* state. Afterwards, in the exploration phase, the ants start to search and harvest resources in the network. The exploration is performed in discrete time steps. A time step ends after all ants perform a move. All ants can perform their moves in parallel, in one time step. An ant in the *empty* state:

- crosses exactly one edge, picked randomly and adjacent to its current location[2]. The probability for an ant a to choose to move from any node $v \in \mathbf{V}$ to any node $w \in \Gamma_v$ is given by:
$$p_{vw}^a = (|\Gamma_v|)^{-1} \tag{4}$$
- searches for *vres* in the new location, and if successful, it will harvest one *vres* and change state to *loaded*;

In the *loaded* state, the ant:

[2]It is worth mentioning, that each ant is keeping a list with the visited nodes, and if the new location of it creates a cycle in its path, all the nodes belonging to the cycle will be removed from the list with visited nodes.

```
%Initialization phase;
set graph 𝒢 and n, m, V, E;
set n_a, n_𝒱, n_t;
set ℒ_a = {} - list with all ants;
for i = 1 : n do
    𝒱_0(v)=n_𝒱;
    for j = 1 : n_a do
        ℒ_a.append(GenerateNewAnt(V_i))
    end
end
%Exploration phase;
for t = 1 : n_t do
    for each ant in ℒ_a do
        % Ant move;
        if ant.state=='empty' then
            % ant chooses next location randomly;
            ant.nextNode=ChooseRandom(Γ_{ant.location});
            ant.visitNodes.Push(ant.location);
            ant.CrossEdge();
            % eliminate cycles in ant path;
            iD=Length(ant.visitedNodes);
            for node in ant.visitedNodes do
                if ant.location == node then
                    iD=IndexOf(node);
                    break;
                end
            end
            ant.visitNodes=ant.visitNodes[1:iD];
            % ant search and harvest vres;
            if ant.location! = ant.home &
            ant.FoundVres()==true then
                ant.HarvestVres();
                ant.state='loaded';
                𝒱_t(ant.location)-=1;
            end
        end
        if ant.state=='loaded' then
            % ant is going back home;
            ant.nextNode=ant.visitNodes.Pop();
            ant.CrossEdge();
            % ant is unloading the vres;
            if ant.location == ant.home then
                ant.state='empty';
                𝒱_t(ant.location)+=1;
            end
        end
    end
end
```

Algorithm 1: Sequential pseudo-code of DANIS

- crosses one edge, in the reverse order of the visited nodes in its last *empty* state;
- If the new location is its home nest, it will unload the *vres* and change its state to *empty*;

Furthermore, each node $v \in \mathbf{V}$ can compute locally its centrality $C_{da}(v)$, using the following formula:

$$C_{da}(v) = \frac{1}{n_t} \sum_{t=1}^{n_t} \mathcal{V}_t(v) \tag{5}$$

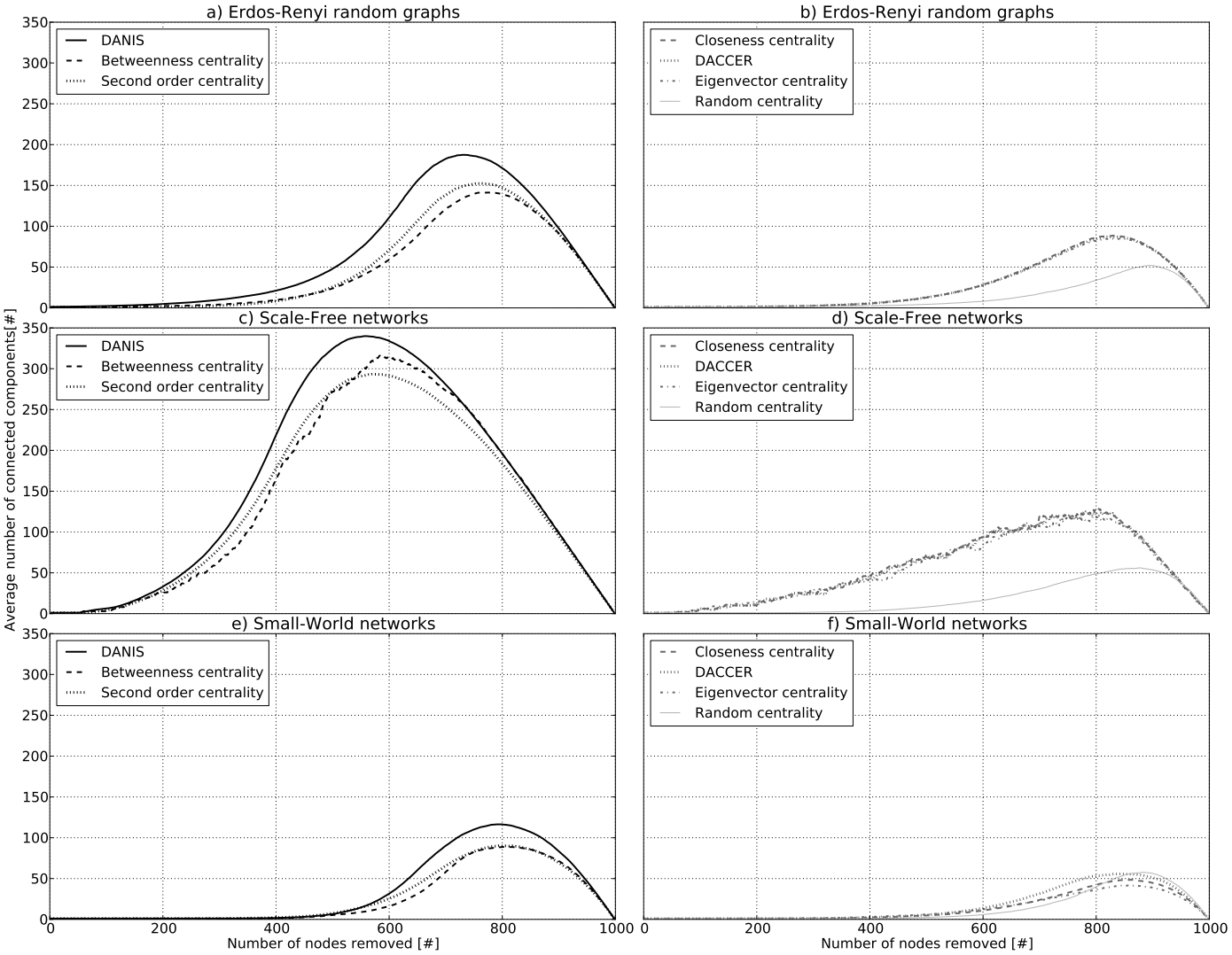

Fig. 1: Assessing DANIS efficiency using node removal procedure. The results are averaged over 100 random generated graphs, for each class of networks under scrutiny. The left column shows the best performers and the right column shows the worst performers, for each specific class.

where n_t represents the total number of time steps took into consideration, and $\mathcal{V}_t(v)$ represents the amount of *vres* which node v has at time step t. To clarify the aforementioned statements, in Algorithm 1, a sequential pseudo-code for DANIS is presented.

IV. Experiments and Results

We assessed the performance of DANIS on three classes of networks: (i) Erdös-Rényi Random Graph; (ii) Scale-Free networks; and (iii) Small-World networks. For each class, we have generated randomly a number of n_g networks (i.e. $n_g = 100$), using the Python library NetworkX [16], with the following characteristics:

- Erdös-Rényi Random Graph: $n = 1000$ nodes, and a connection probability $p = 0.01$, leading to approximately 10000 edges;
- Scale-Free networks: $n = 1000$ nodes, with 5 connections per new node, leading to approximately 10000 edges. The algorithm used to generate them is described in [8];
- Small-World networks: $n = 1000$ nodes, each node being connected with the nearest 6 neighbors, and the probability to add a new edge for each existing edge was set to 0.6. These lead to around 8000 edges in the graph. The algorithm used to generate them is presented in [17].

DANIS was implemented from scratch in Python. For each generated network $n_a = 5$, and $n_{\mathcal{V}} = 300$ *vres*. The evaluation has been made in a step wise fashion, as it is shown further.

A. Efficiency of DANIS

One solution to assess the accuracy of centrality's calculation algorithms, is to remove the vertexes one-by-one and count the connected components. This process starts from the node ranked as most important by the algorithm under scrutiny. A similar approach has been used in [3]. After

each node removal, we checked in the remaining graph, the number of connected components. We expect this number to be bigger if nodes of higher importance are removed. Using, the node removal procedure, we have compared DANIS performance, with the performance of other 6 centralities measures. Three were centralized methods (i.e. Betweenness centrality, Closeness centrality, Eigenvector centrality), two decentralized methods (i.e. Second order centrality, DACCER), and one random removal method. In the case of Second order centrality we stopped the algorithm after the random walk visited 1000000 nodes, as it is recommended in the original paper [3] for network with 1000 nodes. In the case of DANIS, we stopped the algorithm after 40 time steps. In Fig.1 these results, averaged over all 100 networks for each class, are depicted. It is clear, that DANIS outperforms all the other algorithms, in all cases. It is worth highlighting that to achieve these performances, among the decentralized algorithms, DANIS needs 200000 communication messages[3] in the network, while the second performer (i.e. Second order centrality) needs around 5 times more communication messages (i.e. 1000000). Furthermore, independently from the algorithms used, we can observe that it is a clear relation between the robustness[4] of each class of networks, $Robustness(\mathscr{G}_{SF}) < Robustness(\mathscr{G}_{ER}) < Robustness(\mathscr{G}_{SW})$. Table I shows the average number of connected components per algorithm τ per network class out of the whole node removal procedure (1000 nodes, 100 graphs). Formally, this can be written as:

$$\overline{N^{cc}}(\tau) = \frac{1}{n \times n_g} \sum_{r=1}^{n} \sum_{k=1}^{n_g} N_{r,k}^{cc}(\tau) \tag{6}$$

where, τ represents the algorithm used to measure the centrality under scrutiny, and $N_{r,k}^{cc}(\tau)$ represents the number of connected components obtained in the k^{th} generated network, after we removed r nodes sorted accordingly to the importance given by algorithm τ .

TABLE I: Average number of connected components, yielded by the node removal procedure.

	$\mathscr{G}_{ER}$	$\mathscr{G}_{SF}$	$\mathscr{G}_{SW}$
$\overline{N^{cc}}$(DANIS)	**65.72**	**159.28**	**34.15**
$\overline{N^{cc}}$(Betweenness centrality)	45.97	141.29	25.29
$\overline{N^{cc}}$(Second order centrality)	49.42	139.66	27.17
$\overline{N^{cc}}$(Closeness centrality)	27.35	57.32	13.91
$\overline{N^{cc}}$(DACCER)	27.08	56.44	16.80
$\overline{N^{cc}}$(Eigenvector centrality)	26.75	54.77	13.05
$\overline{N^{cc}}$(Random centrality)	12.97	17.35	13.31

B. Learning curve of DANIS

To analyze the evolution of DANIS in time, and to study its behavior, let us note DANIS$_t$ the performance of DANIS after t time steps. In Fig.2 we have plotted $\overline{N^{cc}}(Betweenness) - \overline{N^{cc}}(DANIS_t)$ (i.e. the Y axis), over 100 time steps (i.e. the X axis), for the three classes of networks $\mathscr{G}_{ER}$, $\mathscr{G}_{SF}$, and $\mathscr{G}_{SW}$. In all three cases, DANIS learns very fast and it starts to perform better than Betweenness centrality after around 15 time steps. It is worth highlighting, that this means 75000 communication messages, approximately 13 times faster than Second order centrality. Also, it can be observed that after around 20 time steps, DANIS becomes very stable and it performs constantly better than Betweenness centrality, the learning curve being almost horizontally.

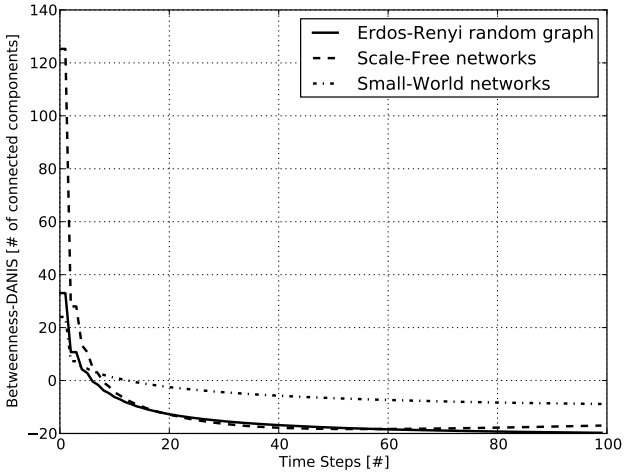

Fig. 2: Learning curve of DANIS over time steps.

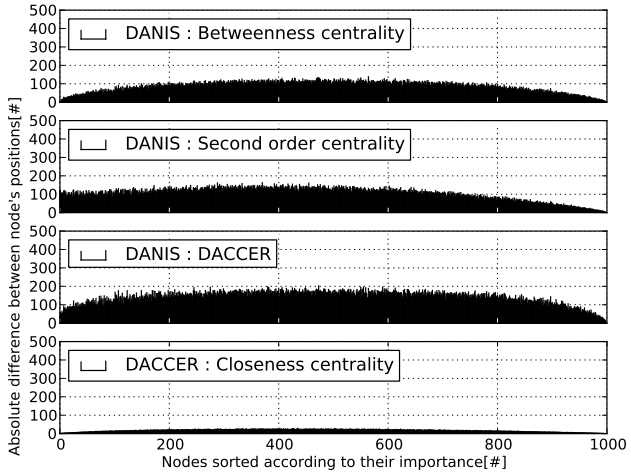

Fig. 3: Difference between centralities on Erdös-Rényi random graphs.

C. Differences between centralities

In the last phase of the experiments, we show that DANIS represents a different metric to assess node's importance in a network. This statement is reflected in Fig. 3, Fig. 4, and Fig. 5 for $\mathscr{G}_{ER}$, $\mathscr{G}_{SF}$, and $\mathscr{G}_{SW}$, respectively. More exactly, each of the plots from the aforementioned figures show the absolute difference between the ranking of a node assigned by the two algorithms compared. As usual, the results are

[3]We consider a communication message when an ant is moving on an edge from one node to the next.

[4]Robustness: The number of connected components remains as small as possible, when the nodes are removed gradually, making the network much more stronger in front of attacks.

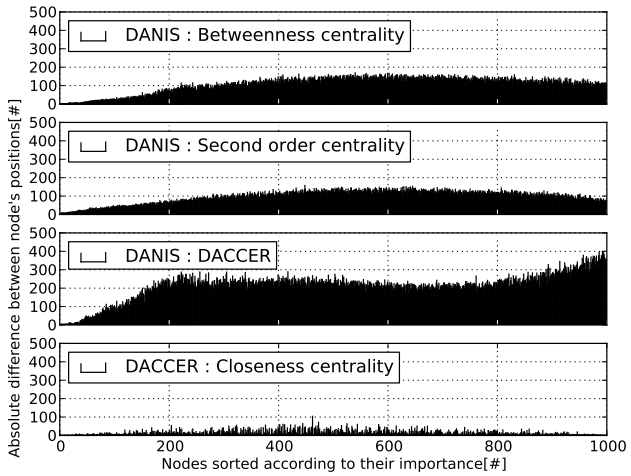

Fig. 4: Difference between centralities on Scale-Free networks.

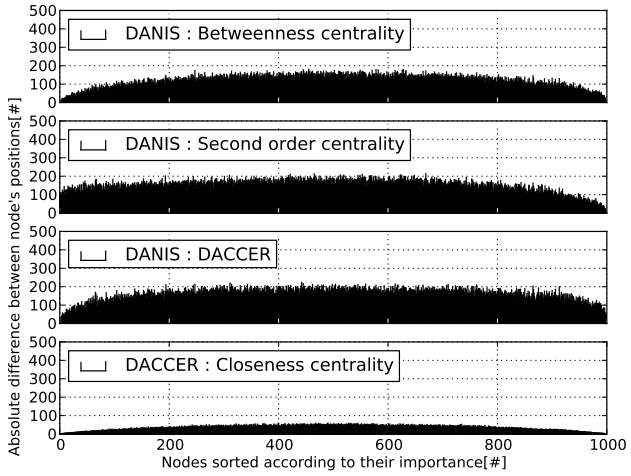

Fig. 5: Difference between centralities on Small-World nets.

averaged over all 100 generated networks for each class. We would like to mention that if both algorithms give exactly the same importance to a node, the absolute difference will be 0. This is clearly reflected in the case of DACCER compared to Closeness centrality, due to the fact that it confirms that DACCER is a decentralized approximation of Closeness centrality, as it is shown in [4]. In the above mentioned figures, we can clearly observe that DANIS is a novel method to measure node's centrality, and not an approximation of any other methods discussed in this paper. DANIS and Betweenness centrality pick mostly the same few most important nodes. These observations together with the evaluation results confirm our intuition, and show that DANIS is capable to evaluate the node's importance better that the state-of-the-art algorithms analyzed in this research.

V. Conclusion

In this paper we introduce a novel algorithm to assess node's centrality in networks, based on the collaborative behavior of decentralized and self-organized swarms, namely DANIS. DANIS has been extensively evaluated in Erdös-Rényi random graphs, Scale-Free networks, and Small-World networks, against state-of-the-art algorithms for centrality estimation such as Betweenness centrality, Second order centrality, Closeness centrality, DACCER and Eigenvector centrality. The results show that DANIS is stable and it outperforms all the other algorithms. As further research directions we would like to study how DANIS behaves in long run if the network topology is changing over time, and to investigate the dynamics of real-world networks using DANIS adaptive capabilities.

References

[1] A. Liotta, "The cognitive NET is coming," *IEEE Spectrum*, vol. 50, no. 8, pp. 26–31, Aug. 2013.

[2] U. Brandes, "On variants of shortest-path betweenness centrality and their generic computation." *Social Networks*, vol. 30, no. 2, pp. 136–145, 2008.

[3] A.-M. Kermarrec, E. L. Merrer, B. Sericola, and G. Trdan, "Second order centrality: Distributed assessment of nodes criticity in complex networks." *Computer Communications*, vol. 34, no. 5, pp. 619–628, 2011.

[4] K. Wehmuth and A. Ziviani, "Daccer: Distributed assessment of the closeness centrality ranking in complex networks," *Computer Networks*, vol. 57, no. 13, pp. 2536–2548, 2013.

[5] P. Bonacich, "Factoring and weighting approaches to status scores and clique identification," *Journal of Mathematical Sociology*, vol. 2, no. 1, pp. 113–120, 1972.

[6] M. E. J. Newman, *Networks: An Introduction*. Oxford University Press, 2010.

[7] P. Erdos and A. Renyi, "On the evolution of random graphs," *Publ. Math. Inst. Hungary. Acad. Sci.*, vol. 5, pp. 17–61, 1960.

[8] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[9] U. Brandes and C. Pich, "Centrality estimation in large networks." *I. J. Bifurcation and Chaos*, vol. 17, no. 7, pp. 2303–2318, 2007.

[10] A. C. B. K. Vendramin, A. Munaretto, M. R. de Biase da Silva Delgado, and A. C. Viana, "Cgrant: a swarm intelligence-based routing protocol for delay tolerant networks." in *GECCO*, T. Soule and J. H. Moore, Eds. ACM, 2012, pp. 33–40.

[11] F. Ducatelle, G. Caro, and L. Gambardella, "Principles and applications of swarm intelligence foradaptive routing in telecommunications networks," *Swarm Intelligence*, vol. 4, no. 3, pp. 173–198, 2010.

[12] K. M. Sim and W. H. Sun, "Multiple ant-colony optimization for network routing," in *First International Symposium on Cyber Worlds, 2002. Proceedings*, 2002, pp. 277–281.

[13] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Scituate, MA, USA: Bradford Company, 2004.

[14] C. Holden, "On the scent of a data trail," *Science*, vol. 278, no. 5342, pp. 1407–, 1997.

[15] G. Exarchakos and N. Antonopoulos, "Cooperative stalking of transient nomadic resources on overlay networks," *Future Gener. Comput. Syst.*, vol. 29, no. 6, p. 14731484, Aug. 2013.

[16] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, Aug. 2008, pp. 11–15.

[17] M. E. J. Newman and D. J. Watts, "Renormalization group analysis of the small-world network model," *Physics Letters A*, vol. 263, no. 4-6, pp. 341–346, 1999.