

# SCIENTIFIC REPORTS



Correction: Publisher Correction

OPEN

## Improved collective influence of finding most influential nodes based on disjoint-set reinsertion

Fengkuangtian Zhu

Identifying vital nodes in complex networks is a critical problem in the field of network theory. To this end, the Collective Influence (CI) algorithm has been introduced and shows high efficiency and scalability in searching for the influential nodes in the optimal percolation model. However, the crucial part of the CI algorithm, reinsertion, has not been significantly investigated or improved upon. In this paper, the author improves the CI algorithm and proposes a new algorithm called Collective-Influence-Disjoint-Set-Reinsertion ( $CI_{DR}$ ) based on disjoint-set reinsertion. Experimental results on 8 datasets with scales of a million nodes and 4 random graph networks demonstrate that the proposed  $CI_{DR}$  algorithm outperforms other algorithms, including Betweenness centrality, Closeness centrality, PageRank centrality, Degree centrality (HDA), Eigenvector centrality, Nonbacktracking centrality and Collective Influence with original reinsertion, in terms of the Robustness metric. Moreover,  $CI_{DR}$  is applied to an international competition on optimal percolation and ultimately ranks in 7th place.

Research on the identification of vital nodes is crucial in computer science, statistical physics and biology applications<sup>1</sup>. Techniques are universally applied in social networks<sup>2–4</sup>, predicting essential proteins<sup>5–7</sup>, quantifying scientific influences<sup>8–10</sup>, detecting financial risks<sup>11–13</sup>, predicting career movements<sup>14,15</sup> and predicting failures with developer networks<sup>16–18</sup>. Considering its importance for application in many fields, the problem of identifying influencers in a network has attracted substantial attention in network analysis.

Researchers have developed numerous measures to evaluate node importance. The most widely used centrality methods include Betweenness centrality<sup>19</sup>, Closeness centrality<sup>20</sup>, PageRank centrality<sup>21</sup>, Degree centrality (HDA)<sup>22</sup>, Eigenvector centrality<sup>23</sup>, and Nonbacktracking centrality<sup>24</sup>. Betweenness centrality<sup>19</sup> is defined to represent a node as the number of shortest paths from all vertices to all other paths that pass through that node. Closeness centrality<sup>20</sup> uses the sum of the length of the shortest paths between the node and all other nodes in a graph as a node's value. PageRank centrality<sup>21</sup> was first proposed by Google to rank websites and works by counting the number and quality of links to a page to determine a rough estimate the importance of a website. Degree centrality (HDA)<sup>22</sup> ranks nodes directly according to the number of connections and recalculates the degree after each removal of the top ranked node. Eigenvector centrality<sup>23</sup> computes the centrality for a node based on the idea that the importance of a node is recursively related to the importance of the nodes pointing to it. A high eigenvector score means that a node is connected to many nodes who themselves have high scores. However, for some graphs, the Eigenvector centrality will produce an echo chamber effect and localization onto a hub. Nonbacktracking centrality<sup>24</sup> modified the standard Eigenvector centrality based on the Nonbacktracking matrix to ignore the reflection mechanism on hubs, therein being asymptotically equivalent to Eigenvector centrality for dense networks and avoiding hub localization on sparse networks.

However, for these methods, the node importance is evaluated by regarding a node as an isolated agent in a non-interacting setting. Consequently, these methods are considered as heuristic methods and fail to provide the optimal solution in the general case of finding a single influential node among multiple spreaders<sup>4,25</sup>. To address the issue, a scalable theoretical framework called the Collective Influence (CI) algorithm, which attempts to find the minimal fraction of nodes that can fragment the network in optimal percolation, was recently proposed<sup>4</sup>. If the influencers are removed in a network, the network will face structural collapse, and a giant connected component  $G$  of the graph will be 0. The CI would improve and maximize the collective influence of multiple influencers, and the accurate optimization objective is highly adaptable for giving an optimal set of spreaders for

Unaffiliated, Shanghai, China. Correspondence and requests for materials should be addressed to F.Z. (email: [zhfkt@hotmail.com](mailto:zhfkt@hotmail.com))

various networks. For networks with millions of nodes, such as massive social media and social networks<sup>22</sup>, CI also performs well in processing centrality efficiently.

The implementation of the CI algorithm contains 2 steps. In the first step, CI calculates the value of each node in the network and removes the node with the highest value according to their importance one by one until the giant component is destroyed. Then, in the second step, CI adds back removed nodes and reconstructs the collapsed network, i.e., reinsertion. Reinsertion<sup>4,22</sup> is the refined post-processing in the CI algorithm and minimizes the giant component  $G$  of the graphs for the target  $G > 0$ . Although CI has already demonstrated its efficiency in searching for the potential influential nodes in the optimal percolation model, the reinsertion step in CI has rarely been discussed. Until now, the optimal percolation model only addressed the issue of dismantling networks in the first step of CI. The second procedure, reinsertion, is not designed to be optimal, which leads to the fact that the former optimal percolation model is unable to achieve optimal results. Therefore, it is necessary to address this issue by designing a better reinsertion step.

Robustness<sup>26</sup> is a recently proposed measure for quantifying the performance of methods for ranking nodes. This paper improves the reinsertion method with respect to the Robustness metric to find the most influential nodes in CI, and it proposes a new algorithm named Collective-Influence-Disjoint-Set-Reinsertion ( $CI_{DR}$ ).  $CI_{DR}$  mainly employs disjoint sets<sup>27</sup> as the data structure to optimize reinsertion in the CI algorithm and reorder the removed nodes into a new sequence.

The proposed  $CI_{DR}$  method is verified in the International Competition of optimal percolation<sup>28</sup> and ultimately ranks in 7th place. The competition adopts the Robustness metric as the scoring criteria and provides 4 real networks from different fields, i.e., autonomous system networks, Internet networks, road networks and social networks, and 4 classical artificial networks (8 datasets in total). The node counts of these networks range from 0.4 million to 2 million. Therefore, the competition network benchmark quite representative overall. The results of the experiments indicate that the proposed  $CI_{DR}$  method outperforms the other 7 methods on 8 competition datasets. The methods include Betweenness centrality<sup>19</sup>, Closeness centrality<sup>20</sup>, PageRank centrality<sup>21</sup>, Degree (HDA) centrality<sup>22</sup>, Eigenvector centrality<sup>23</sup>, Nonbacktracking centrality<sup>24</sup> and Collective Influence with original reinsertion<sup>22</sup> as comparison algorithms.

A total of 4 extra random graphs in the ER model generated locally are also utilized to verify  $CI_{DR}$ . The results on the 4 random graphs show that  $CI_{DR}$  is also better than the other methods listed in the paper, similar to the results on the 8 above competition datasets.

## Results

**Difference between reinsertion in CI and Collective-Influence-Disjoint-Set-Reinsertion ( $CI_{DR}$ ).** The CI algorithm contains 2 steps: removing nodes and reinsertion. For the removing node step, CI calculates the value of  $node_i$  in Formula 1 and removes the node with the highest value.

$$CI(i) = (k_i - 1) \sum_{j \in \delta B(i,l)} (k_j - 1) \quad (1)$$

$k_i$  is defined as the degree of  $node_i$ ,  $\delta B(i, l)$  is the frontier of the ball centered on  $node_i$  with Radius  $l$ , which refers to the shortest path  $l$  from frontier nodes to  $node_i$ <sup>4</sup>. The newly proposed  $CI_{DR}$  method also calculates the value of each node following Formula 1, which is the same as in CI. The difference between CI and  $CI_{DR}$  is that they implement different strategies in the reinsertion step. In CI, the original reinsertion step is invoked<sup>22</sup> in Algorithm 1 after the networks are broken down into many pieces through the process of removing nodes. An initial collapsed graph  $G_c$  is generated after CI removes nodes from the graph  $G$ , and then, reinsertion selects the removed nodes to reconstruct the collapsed graph  $G_c$ .

---

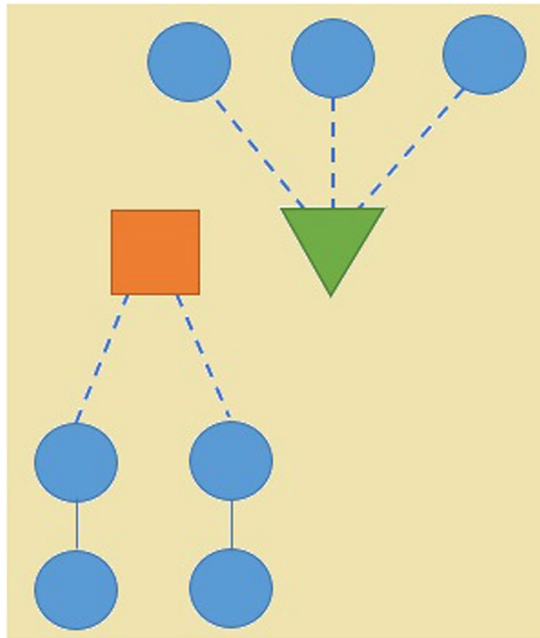
**Algorithm 1.** The overall flow of the original reinsertion process in the CI algorithm.

---

- 1: The original reinsertion method utilizes brute-force graph traversal of an adjacency list or adjacency matrix to label the indices of connected components in the collapsed graph  $G_c$ .
  - 2: For each removed  $node_i$ , if reinserted, the original reinsertion will retrieve the indices of the connected components of  $n$  neighboring nodes around  $node_i$  labeled in Step 1. The indices of the connected components of the  $n$  neighboring nodes around  $node_i$  are denoted by the set  $I_i : I_{i1}, I_{i2} \dots, I_{in}$ .
  - 3: Check whether there are duplicate indices of connected components in the set  $I_i$  and remove the repeated indices. A new set containing distinct indices  $DI_i$  is  $DI_{i1}, DI_{i2} \dots, DI_{in}$ .
  - 4: Obtain the count of the set  $DI_i$  as  $M_i$ . This refers to the number of rejoined clusters if the removed  $node_i$  is reinserted.
  - 5: Search for the smallest  $M_{min}$  value of  $node_{min}$  among all removed node values  $M_i$ .
  - 6: Original reinsertion will reinsert the corresponding  $node_{min}$  into the network  $G_c$ . When reinsertion is implemented, a finite fraction of nodes are added back at each step.
  - 7: Loop to Step 1 and repeat to search for the next reinserted node until all removed nodes are consumed and reinserted into networks.
- 

For the reinsertion step in the improved  $CI_{DR}$ , 2 main enhancements are proposed in this paper:

- $CI_{DR}$  implements disjoint sets as the data structure<sup>27</sup> to store the indices of the connected components during reinsertion for a collapsed graph.
- $CI_{DR}$  considers the rejoined node count instead of the number of rejoined clusters to decide which node will be reinserted.



**Figure 1.** Different choices of candidate reinserted nodes decided by the original reinsertion and  $CI_{DR}$  methods. The original reinsertion method in CI reinserts the square node, and  $CI_{DR}$  reinserts the triangle node.

In the original reinsertion in Algorithm 1, CI utilizes brute-force graph traversal to label the indices of the connected components in a dismantled network (Step 1). The time cost is high since the labeling operation will be executed multiple times until all removed nodes are reinserted. In particular, when a dismantled network has nearly reached the completion of the reinsertion process and most of the removed nodes are reinserted, the labeling operation must build up from nothing every time. Considering that deciding which node will be reinserted is performed several times in the original reinsertion process, the information of the reinsertion for each iteration can be reserved and prepared for the next round of decisions on which removed node will be reinserted.

The first enhancement of  $CI_{DR}$  implements disjoint sets to optimize the data structure to reduce the computational resource consumption. A disjoint set<sup>27</sup> is a tree structure, where each node stores a pointer to the parent node. If the parent pointer of a node points to itself, this node is the root of a tree and is the representative index of its cluster. In Algorithm 1, the index information of the connected components in the reconstructed graph is abandoned at the end of each iteration for updating indices. Using the disjoint-set data structure, it is possible to maintain the indices of the connected components for a collapsed network when the dismantled nodes are reinserted into the graphs. The disjoint-set data structure provides 2 nearly constant-time operations. The first operation is called the *Find* operation, which determines which indices of connected components the current nodes stay in. The second operation is the *Union* operation, which merges several clusters into one.

The *Find* operation locates which connected components a node belongs to. The operation can follow the parent node continuously in a tree of a cluster until it finds the root node, which denotes the index of a connected component. The *Find* operation is utilized to replace Step 2 in Algorithm 1 and is capable of retrieving the index set  $I_i$  of the connected components of the neighboring nodes around  $node_i$ .

The *Union* operation merges clusters to which 2 nodes belong into one connected component. This operation uses the *Find* operation to determine the roots of the trees. If the roots of 2 nodes are distinct, the trees are combined by attaching the root of one to the root of the other node. When newly removed nodes are reinserted, the *Union* operation is capable of preserving the index information of the connected components in the iteration when updating indices continuously.

Several optimization methods on disjoint sets, such as *Path Compression* and *Union by Size*, are applied in the implementation to improve the *Find* and *Union* operations<sup>29</sup>. *Path Compression* flattens the structure of the tree by making every node point to the root when the first *Find* operation is invoked on the tree. This will speed up and decrease the complexity of future *Find* operations. *Union by Size* means that the *Union* operation attaches the tree with fewer nodes to the root of the nodes containing more elements. This is also another method for flattening the structure of the tree. The size of the cluster is stored in the root node of a tree, and the new size of the cluster following the *Union* operation is equal to the sum of the sizes of the root nodes of the original trees.

The computational complexity of both the *Find* and *Union* operations is  $O(\text{inverse\_oka}(n))$  when the *Path Compression* and *Union by Size* optimization methods are utilized, where  $\text{inverse\_oka}(n)$  represents the inverse Ackermann function. The inverse Ackermann function contains a value  $\text{inverse\_oka}(n) < 5$  for any very large value of  $n$  that can be written in this physical universe. Therefore,  $\text{inverse\_oka}(n)$  in the *Find* and *Union* operations is optimal and can essentially be regarded as constant time<sup>29,30</sup>. In the experiment analysis section below, the statistical results also show that utilizing the disjoint-set data structure in  $CI_{DR}$  achieves greater efficiency and is faster than the original reinsertion algorithm in CI.

Network	model1	model2	model3	model4	real1	real2	real3	real4
Number of nodes	1039722	1083568	997663	1001733	1694616	1957027	426485	855802

**Table 1.** The 8 network datasets in the DataCastle Master Competition.

Radius 0	model 1	model 2	model 3	model 4	real 1	real 2	real 3	real 4	total
CI Without Reinsertion	0.2279	0.1914	0.3794	0.1446	0.0493	0.0689	0.1102	0.0922	1.2638
CI with Original Reinsertion	0.2130	0.1783	0.3488	0.1304	0.0459	0.0918	0.1030	0.0751	1.1863
$CI_{DR}$	0.2100	0.1744	0.3603	0.1175	0.0315	0.0069	0.0978	0.0417	1.0403
Radius 1	model 1	model 2	model 3	model 4	real 1	real 2	real 3	real 4	total
CI Without Reinsertion	0.2253	0.1878	0.3809	0.1434	0.0523	0.1069	0.1118	0.1024	1.3108
CI with Original Reinsertion	0.2079	0.1729	0.3459	0.1258	0.0407	0.0545	0.0969	0.0654	1.1099
$CI_{DR}$	0.2104	0.1743	0.3656	0.1152	0.0318	0.0046	0.0968	0.0421	1.0409
Radius 2	model 1	model 2	model 3	model 4	real 1	real 2	real 3	real 4	total
CI Without Reinsertion	0.2239	0.1867	0.3777	0.1317	0.0482	0.0941	0.1085	0.0962	1.2671
CI with Original Reinsertion	0.2083	0.1732	0.3446	0.1189	0.0387	0.0417	0.0955	0.0492	1.0701
$CI_{DR}$	0.2107	0.1739	0.3585	0.1148	0.0303	0.0039	0.0954	0.0370	1.0246

**Table 2.** Robustness score of CI without Reinsertion, with Original Reinsertion and  $CI_{DR}$  on 8 competition datasets for radii of 0, 1, and 2.

For the second enhancement,  $CI_{DR}$  considers the number of rejoined nodes instead of the number of rejoined clusters when deciding which removed nodes will be reinserted. This improvement is applied in Step 4 in Algorithm 1. The purpose of the modification is to enhance the final Robustness score of the original reinsertion operation in the CI algorithm. The original reinsertion operation adds back the nodes that rejoin the smallest number of clusters. Nevertheless, the method does not consider the smallest node count of the rejoined clusters globally. In contrast to the number of rejoined clusters, the information about the rejoined node counts is more representative for a connected component. Because the first enhancement implements the disjoint-set data structure and because the *Union by Size* optimization is enabled, the node count of each connected component is stored in the root node of the corresponding tree in the disjoint set. The smallest node count of rejoined connected components can be conveniently selected from all candidate nodes.

Figure 1 is an example of different choices of the candidate reinserted nodes decided by the original reinsertion and  $CI_{DR}$  methods. Round nodes have been in the collapsed network, and there are 2 candidate nodes to be reinserted: the square node and the triangle node. If the original reinsertion method in CI is applied, it will reinsert the square node because the number of rejoined clusters is 2; fewer than 3 clusters are reinserted by the triangle node. If  $CI_{DR}$  is exploited, it will reinsert the triangle node because there are 3 rejoined nodes, and fewer than 4 rejoined nodes are reinserted by the square nodes.  $CI_{DR}$  more strongly considers the global impact of the rejoined nodes on the collapsed network compared with the original reinsertion process. In conclusion, the  $CI_{DR}$  algorithm with the first and second enhancements for the reinsertion process is shown in Algorithm 2.

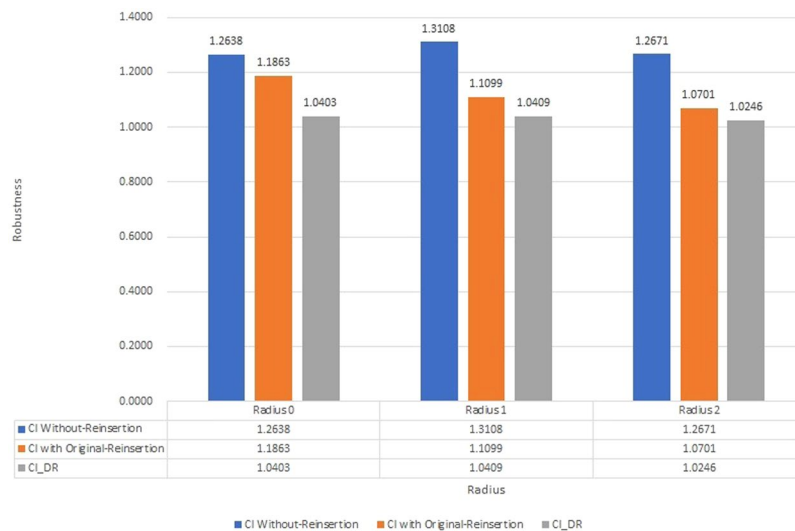
---

**Algorithm 2.** Collective-Influence-Disjoint-Set-Reinsertion ( $CI_{DR}$ ).

---

- 1: An initial collapsed graph  $G_c$  is generated after  $CI_{DR}$  removes nodes from the original graph  $G$  by calculating the highest value of Formula 1 one by one. This step is the same as the node removal step in CI.  $CI_{DR}$  will create the corresponding disjoint-set data structure  $DS_G$  until the giant component is destroyed.
  - 2: Invoke improved reinsertion and select removed nodes to reconstruct the graph  $G_c$ :
    - a: For each removed  $node_i$ , if reinserted once,  $CI_{DR}$  will retrieve the indices of the connected components of  $n$  neighboring nodes around  $node_i$  by the *Find* operation on the disjoint-set data structure  $DS_G$ . The indices of the connected components of  $n$  neighboring nodes around  $node_i$  are given by the set  $I_i : I_{i1}, I_{i2}, \dots, I_{in}$ .
    - b: Check whether there are duplicate indices of connected components in the set  $I_i$  and remove the repeated indices. A new set containing distinct indices  $DI_i$  is  $DI_{i1}, DI_{i2}, \dots, DI_{im}$ .
    - c: Obtain node counts of  $m$  distinct connected components  $count[DI_i]$  from the root node indices in the trees. Sum  $count[DI_{i1}], count[DI_{i2}], \dots, count[DI_{im}]$  as  $S_i$  to achieve the total number of rejoined nodes if the removed  $node_i$  is reinserted.
    - d: Search for the smallest  $S_{min}$  value of  $node_{min}$  among all removed node values  $S_i$ .
    - e:  $CI_{DR}$  will reinsert the corresponding  $node_{min}$  into the network  $G_c$  via the *Union* operation. The indices of the connected components in the disjoint-set data structure  $DS_G$  are also updated. When reinsertion is implemented, the finite fraction of nodes are added back at each step.
    - f: Repeat searching for the next reinserted node in Step 2 until all removed nodes are consumed and reinserted into the networks.
- 

**Experiments and comparison of different methods on 8 datasets provide by DataCastle Master Competition.** In this subsection, several centrality methods are verified on 8 datasets provided by the DataCastle Master Competition<sup>28</sup>. The task of the competition is a generic challenge identifying vital nodes in



**Figure 2.** Total Robustness score of CI Without Reinsertion, with Original Reinsertion and  $CI_{DR}$  on 8 competition datasets for radii of 0, 1, and 2.

Radius 0	model	model 2	model 3	model 4	real 1	real 2	real 3	real 4
Original Reinsertion	166s	140s	283s	103s	686s	82s	477s	153s
$CI_{DR}$	129s	101s	195s	78s	183s	85s	152s	129s
Radius 1	model 1	model 2	model 3	model 4	real 1	real 2	real 3	real 4
Original Reinsertion	157s	132s	282s	99s	482s	56s	458s	152s
$CI_{DR}$	103s	89s	190s	73s	167s	50s	167s	111s
Radius 2	model 1	model 2	model 3	model 4	real 1	real 2	real 3	real 4
Original Reinsertion	165s	119s	276s	91s	381s	42s	262s	104s
$CI_{DR}$	40s	55s	201s	34s	105s	36s	54s	98s

**Table 3.** Time consumption of Original Reinsertion and  $CI_{DR}$  excluding the step of removing nodes on 8 competition datasets.

networks that are important for sustaining connectivity. The competition provides 4 real networks from different fields, i.e., autonomous system networks, Internet networks, road networks and social networks, and 4 classical artificial networks (for a total of 8 datasets). These networks each include 0.4 million to 2 million nodes, and all networks are considered undirected networks. Table 1 reflects the network name and corresponding number of nodes.

Robustness<sup>26</sup> is utilized as the scoring criterion in the competition. It is introduced to quantify the performance of the methods for ranking nodes. For the calculation of the Robustness score, refer to Formula 2.

$$R = \frac{1}{N} \sum_{i=1}^N \delta \left( \frac{i}{n} \right) \quad (2)$$

The parameter  $p$  is defined as the proportion of removed nodes.  $\delta$  is the size of the giant component of the remaining networks in proportion after removing a proportion  $p$  of the nodes. The  $\delta$ - $p$  curve can be derived from plotting  $p$  on the x-axis and  $\delta$  on the y-axis. The Robustness is defined as the area under the  $\delta$ - $p$  curve.  $\delta \left( \frac{i}{n} \right)$  is the size of the giant component after removing  $p = \frac{i}{n}$  of the nodes from a network<sup>28</sup>.

Generally, the goal of an algorithm that finds the most influential nodes is to give a ranked list of nodes according to their importance, where the top-ranked nodes will have greater importance. Nodes can be removed from a network according to the ranking list. The removal operation breaks down the network into many disconnected pieces. If the size of the giant component is calculated after the removal of each node, the ratio of the giant component will ultimately go to 0. Therefore, a better algorithm for ranking nodes will dismantle networks sooner and produce better Robustness scores.

The Robustness under CI without Reinsertion, with Original Reinsertion and  $CI_{DR}$  on 8 competition datasets is presented in Table 2 and Fig. 2. CI without Reinsertion refers to the case in which CI only invokes the process of removing nodes and does not reinsert nodes into networks. CI with Original Reinsertion refers to the case with the node removal step and original reinsertion in Algorithm 1. For various similar CI algorithms, the radius is the required input parameter for the node removal step. A larger radius will optimize removing steps and produce a smaller set of minimal influential nodes when dismantling networks. As a trade-off, the step of removing nodes



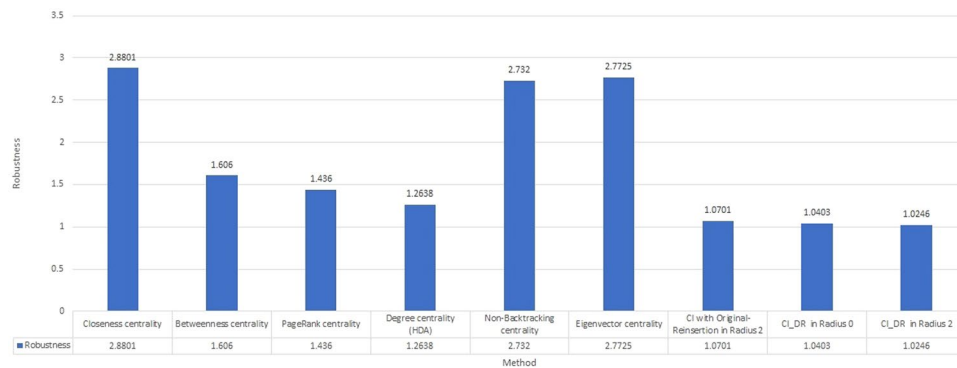


**Figure 3.** Time consumption (seconds) of Original Reinsertion and  $CI_{DR}$  excluding the step of removing nodes on 8 competition datasets. (a) Radius = 0. (b) Radius = 1. (c) Radius = 2.

	model 1	model 2	model 3	model 4	real 1	real 2	real 3	real 4	Total
Betweenness centrality	0.3125	0.2678	0.4484	0.1952	0.1101	0.0064	0.1582	0.1076	1.6060
Closeness centrality	0.4140	0.3765	0.4624	0.3717	0.2495	0.4454	0.2738	0.2868	2.8801
PageRank centrality	0.2435	0.2043	0.4267	0.1469	0.0522	0.1364	0.1430	0.0832	1.436
Degree centrality (HDA)	0.2279	0.1914	0.3794	0.1446	0.0493	0.0689	0.1102	0.0922	1.2638
Eigenvector centrality	0.4228	0.4163	0.4624	0.3938	0.2718	0.2222	0.2892	0.2939	2.7725
Nonbacktracking centrality	0.4142	0.3850	0.4624	0.3757	0.2715	0.2443	0.2885	0.2905	2.7320

**Table 4.** Robustness of different heuristic algorithms, including Betweenness centrality, Closeness centrality, PageRank centrality, Degree centrality (HDA), Eigenvector centrality and Nonbacktracking centrality, on 8 competition datasets.

will cost more computing resources and take longer. Meanwhile, a larger radius will speed up reinsertion because fewer dismantled nodes are reinserted. The result of using different input radii of 0, 1, and 2 for the various CI methods is also shown in Table 2 and Fig. 2.



**Figure 4.** Total Robustness of different heuristic algorithms, including Betweenness centrality, Closeness centrality, PageRank centrality, Degree centrality (HDA), Eigenvector centrality and Nonbacktracking centrality, on 8 competition datasets

Network	Random 0	Random 1	Random 2	Random 3
Number of nodes	$10^5$	$10^5$	$10^6$	$10^6$
Mean degree	2	3	2	3

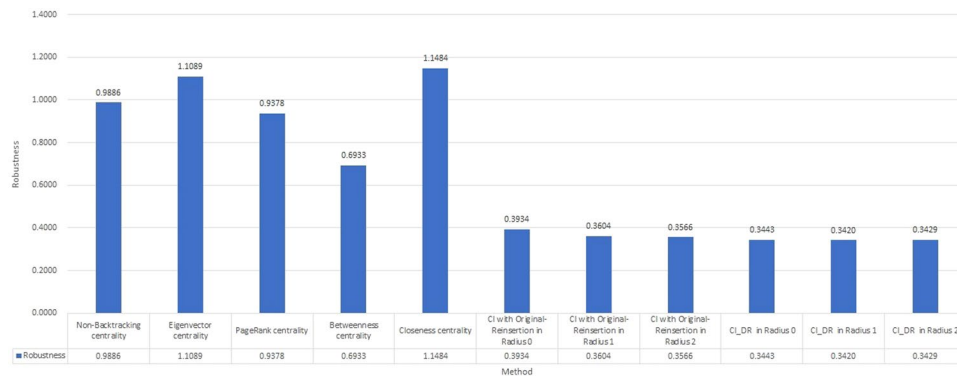
**Table 5.** 4 network datasets for randomly generated graphs in the ER model.

Network	Random 0	Random 1	Random 2	Random 3	Total
Betweenness centrality	0.1169	0.2232	0.1209	0.2323	0.6933
Closeness centrality	0.2687	0.3045	0.2685	0.3068	1.1484
PageRank centrality	0.1740	0.2962	0.1736	0.2941	0.9378
Eigenvector centrality	0.2482	0.3036	0.2501	0.3070	1.1089
Nonbacktracking centrality	0.2001	0.2947	0.1971	0.2967	0.9886
CI with Original Reinsertion (Radius = 0)	0.0654	0.1326	0.0628	0.1326	0.3934
CI with Original Reinsertion (Radius = 1)	0.0560	0.1252	0.0542	0.1250	0.3604
CI with Original Reinsertion (Radius = 2)	0.0548	0.1248	0.0523	0.1246	0.3566
CI <sub>DR</sub> (Radius = 0)	0.0492	0.1234	0.0480	0.1237	<b>0.3443</b>
CI <sub>DR</sub> (Radius = 1)	0.0489	0.1218	0.0490	0.1224	<b>0.3420</b>
CI <sub>DR</sub> (Radius = 2)	0.0484	0.1214	0.0480	0.1251	<b>0.3429</b>

**Table 6.** Robustness value of CI with Original Reinsertion, CI<sub>DR</sub> and other heuristic algorithms, including Betweenness centrality, Closeness centrality, PageRank centrality, Eigenvector centrality and Nonbacktracking centrality, on 4 random graphs.

Table 2 and Fig. 2 show that the total Robustness (the lower, the better) of CI<sub>DR</sub> on the 8 datasets is better than in the cases without Reinsertion and with Original Reinsertion. This is because in the reinsertion process, the rejoined node count is more representative in vital nodes than the number of rejoined clusters in the Robustness metric. For a radius of 0, the total Robustness score under CI with Original Reinsertion of 1.1863 decreases by 12% to 1.0403 for CI<sub>DR</sub>. For a radius of 1, the total Robustness score of 1.1099 for CI decreases by 6% to 1.0409 for CI<sub>DR</sub>. For a radius of 2, the total Robustness score of 1.0701 decreases 4% to 1.0246 for CI<sub>DR</sub>. For each individual dataset, CI<sub>DR</sub> performs better than CI in terms of Robustness in 7 of the 8 networks when a radius of 0 is used in CI and CI<sub>DR</sub>. CI<sub>DR</sub> ranks second to CI only in the model 3 network. When a radius of 1 and a radius of 2 are adopted in CI and CI<sub>DR</sub>, CI<sub>DR</sub> obtains a better score than CI on 5 of the 8 networks. CI<sub>DR</sub> ranks second behind CI in the model 1, model 2 and model 3 networks and obtains nearly the same and best result.

For different radii as input parameters, the total Robustness values in CI<sub>DR</sub> are all better than those with Original Reinsertion. Even for the case when the radius is 0 and the process of removing nodes degenerates into that of Degree centrality (HDA)<sup>22</sup>, CI<sub>DR</sub> is capable of achieving a considerably better score of 1.0403 compared with the previous best result of 1.0701 under Original Reinsertion with a radius of 2. For the smaller radius, CI<sub>DR</sub> exploits potential performance increases in terms of the Robustness metric, in contrast to Original Reinsertion. Therefore, there is no need to set a higher radius, which increases the complexity of removing nodes in CI. A lower radius is able to achieve nearly the same results in CI<sub>DR</sub>.



**Figure 5.** Total Robustness value of CI with Original Reinsertion,  $CI_{DR}$  and other heuristic algorithms, including Betweenness centrality, Closeness centrality, PageRank centrality, Eigenvector centrality and Nonbacktracking centrality, on 4 random graphs.

In Table 3 and Fig. 3, the time consumption of Original Reinsertion and  $CI_{DR}$  excluding removing nodes on 8 competition datasets is presented. The datasets are verified on the same machine with a 4-core CPU (Intel Xeon E5-2667v4 Broadwell 3.2 GHz) with 8 GB of memory concurrently. For most cases with different radii, the statistics show that  $CI_{DR}$  is better in terms of speed than Original Reinsertion, excluding the node removal steps, in CI. For the real 3 dataset with a radius of 2, the time consumption can be reduced 79.3% from 262 s to 54 s. The statistics evidence that implementing the disjoint-set data structure in  $CI_{DR}$  is more efficient than the Original Reinsertion algorithm.

Heuristic algorithms, including Betweenness centrality<sup>19</sup>, Closeness centrality<sup>20</sup>, PageRank centrality<sup>21</sup>, Degree centrality (HDA)<sup>22</sup>, Eigenvector centrality<sup>23</sup> and Nonbacktracking centrality<sup>24</sup>, are verified on these datasets as competitors in Table 4 and Fig. 4. The Robustness values of these heuristic algorithms are all worse than CI and  $CI_{DR}$  for the 8 competition networks. The worst value of *Closeness centrality* is only 2.88.

The recently proposed Nonbacktracking centrality<sup>24</sup> is also verified on these 8 competition datasets. Nonbacktracking centrality was introduced by Newman *et al.* and modified from the standard Eigenvector centrality based on the Hashimoto or Nonbacktracking matrix<sup>31–33</sup>. Nonbacktracking centrality is very similar to Eigenvector centrality, where the main improvement is to ignore the echo chamber effect producing localization on a hub. This is asymptotically equivalent to Eigenvector centrality for dense networks and avoids the hub localization on sparse networks introduced by Eigenvector centrality. Therefore, the performance of Nonbacktracking centrality in dense networks will be highly similar to Eigenvector centrality. From Table 4 and Fig. 4, the statistics show that corresponding scores of 2.7725 and 2.7320 for Eigenvector centrality and NonBacktracking centrality, respectively, are quite similar. Both Nonbacktracking centrality and Eigenvector centrality are not superior to CI and the proposed  $CI_{DR}$  in terms of Robustness for the 8 competition datasets.

**Experiments and comparison of different methods on 4 randomly generated graphs.** In addition to the 8 above-mentioned competition datasets, 4 random graph networks in the ER model<sup>34</sup> are also adopted as additional test cases. Table 5 shows the information about the number of nodes and mean degree for each graph. The Robustness values under CI with Original Reinsertion,  $CI_{DR}$  and other heuristic algorithms, including Betweenness centrality, Closeness centrality, PageRank centrality, Eigenvector centrality and Nonbacktracking centrality, are presented in Table 6 and Fig. 5 for 4 random graphs.

The total Robustness of  $CI_{DR}$  is 0.3443, 0.3420 and 0.3429 for radii of 0, 1, and 2 on 4 randomly generated graphs, which all outperform the other listed centrality methods. For each individual dataset, the Robustness of  $CI_{DR}$  also ranks 1st out of all applied methods. For a radius of 0,  $CI_{DR}$  performs better in terms of Robustness than CI with Original Reinsertion for a radius of 2. The same result as that on the 8 above-mentioned competition datasets that a lower radius under  $CI_{DR}$  outperforms a higher radius under CI with Original Reinsertion is found.

The NonBacktracking centrality achieves a score of 0.9886, which is slightly better than the score of 1.1089 of the Eigenvector centrality; this is because the former method is based on the latter method. However, a score of 0.9886 is unable to compete with CI and  $CI_{DR}$ , similar to the results on the 8 above competition datasets.

## Discussion

For 8 competition datasets and 4 local randomly generated graphs under the ER model, the best overall result from the previous algorithms is CI with Original Reinsertion for a radius of 2. After the newly proposed algorithm  $CI_{DR}$  is applied, even  $CI_{DR}$  employing a radius of 0 (degenerate to HDA) is capable of achieving a better result. This indicator shows that the proposed disjoint-set reinsertion in  $CI_{DR}$  is able to achieve better Robustness compared to Original Reinsertion. The recently proposed Nonbacktracking centrality and the other above-mentioned algorithms are also unable to outperform  $CI_{DR}$  in terms of Robustness.

CI with Original Reinsertion uses the number of rejoined clusters to decide which node will be reinserted. On the other hand,  $CI_{DR}$  considers the rejoined node count in the second proposed enhancement. Nevertheless, CI with Original Reinsertion and  $CI_{DR}$  implement different methods; both methods attempt to obtain a score that is capable of representing *node<sub>i</sub>*. Therefore, a reinsertion framework derived from  $CI_{DR}$  can be extended to a more general model. The Generic Disjoint-set Reinsertion Framework (GDRF) is proposed in Algorithm 3 as a general method for describing the process of disjoint-set reinsertion.



---

**Algorithm 3.** Algorithm of reinsertion with kernel in a more generic framework: Generic Disjoint-set Reinsertion Framework (GDRF).
 

---

- 1: Create the disjoint-set data structure  $DS_G$  for the corresponding collapsed graph  $G_c$ .
  - 2: For each removed  $node_i$ , if reinserted once, GDRF will retrieve the indices of the connected components of  $n$  neighboring nodes of  $node_i$  by the *Find* operation in the disjoint-set  $DS_G$ . The indices of the connected components of the  $n$  neighboring nodes around  $node_i$  are given by the set  $I_i : I_{i1}, I_{i2} \dots, I_{in}$ .
  - 3: Check whether there are duplicate indices of connected components in the set  $I_i$  and remove the repeated indices. A new set containing distinct indices  $DI_i$  is given as  $DI_{i1}, DI_{i2} \dots, DI_{im}$ .
  - 4: Obtain the score  $K_i$  by a kernel that can represent  $node_i$ .
  - 5: Search for the smallest  $K_{min}$  of  $node_{min}$  among all removed node values  $K_i$ .
  - 6: GDRF will reinsert the corresponding  $node_{min}$  into the network  $G_c$  by the *Union* operation. The indices of the connected components in the disjoint-set data structure  $DS_G$  are also updated. When reinsertion is implemented, a finite fraction of nodes are added back at each step.
  - 7: Loop to Step 1 and repeat to search for the next reinserted node until all removed nodes are consumed and reinserted into networks.
- 

Step 4 in Algorithm 3 retrieves the score  $K_i$  by a specific kernel indicating  $node_i$ , and the other steps in Algorithm 3 are the same as in the reinsertion in the  $CI_{DR}$  Algorithm 2. GDRF employing the *number Of Clusters* kernel (Algorithm 4) and the *number Of Nodes* kernel (Algorithm 5) is able to achieve the same Robustness score as CI with Original Reinsertion and  $CI_{DR}$ , respectively.

---

**Algorithm 4.** CI with Original Reinsertion: number of Clusters kernel used to obtain the score  $K_i$  representing  $node_i$ .
 

---

- 1: Obtain the count of set  $DI_i$  as  $M_i$ . It refers to the number of rejoined clusters if the removed  $node_i$  is reinserted.
  - 2: return  $M_i$  as  $K_i$
- 

---

**Algorithm 5.**  $CI_{DR}$ : *numberOfNodes* kernel used to obtain the score  $K_i$  representing  $node_i$ .
 

---

- 1: Obtain the node counts of  $m$  distinct connected components  $count[DI_i]$  from root node indices in trees. Add up  $count[DI_{i1}]$ ,  $count[DI_{i2}]$ , ...,  $count[DI_{im}]$  as  $S_i$  to obtain the total number of rejoined nodes if removed  $node_i$  is reinserted.
  - 2: return  $S_i$  as  $K_i$
- 

Since the reinsertion of independent post-processing can be combined with any network dismantling method to obtain an improved Robustness score, the greater potential of GDRF can be investigated. For the task of finding the most influential nodes in complex networks, the *number Of Clusters* kernel and the *number Of Nodes* kernel are implemented to reinsert the nodes with less importance in priority.

In future work, more kernels can be studied to investigate whether GDRF can be applied to other issues. For instance, reconstructing damaged networks is also a widely studied field, and various repair strategies have been presented to repair collapsed networks<sup>35–39</sup>. A new kernel for GDRF, which would be designed to reinsert the nodes with greater importance in priority, can be developed for the task of recovering attacked networks as soon as possible. The *Recover Nodes* kernel in Algorithm 6 combined with GDRF is an example of selecting the nodes with greater importance in priority.  $S_i$  represents the total number of rejoined clusters if the removed  $node_i$  is reinserted. A larger  $S_i$  means that recovering  $node_i$  would connect and repair more connected components in the reinsertion process. Compared with the random reinsertion of removed nodes, the *recover Nodes* kernel tends to reinsert nodes combining with more connected components, which means that recovering a network to certain giant components  $G > 0$  would need fewer reinserted nodes. Since GDRF will search for the minimum  $K_{min}$  value of  $node_{min}$  among all removed node values  $K_i$ , Algorithm 6 would return the reciprocal of  $S_i$  as  $K_i$ , and  $node_i$  with larger values of  $S_i$  would be reinserted in priority.

---

**Algorithm 6.** Recovering attacked network: *recoverNodes* kernel used to obtain the score  $K_i$  representing  $node_i$ .
 

---

- 1: Call *numberOfClusters* kernel used to obtain the returned value as  $S_i$  for  $node_i$ .
  - 2: return  $1 / S_i$  as  $K_i$
- 

The *RecoverNodes* kernel is only suitable when the nodes in a network, instead of the edges, are attacked. The *RecoverNodes* kernel will also not modify the original network topology after the recovery process. Because this paper mainly focuses on the influential nodes and not repairing attacked networks, additional research on the performance of the *recoverNodes* kernel compared with previous algorithms can be conducted in future studies.

## Methods

As mentioned above,  $CI_{DR}$  calculates the value of each node of a network using Formula 1 and removes the nodes with the highest value. In particular, if the radius is set to 0, the step of removing nodes in CI and  $CI_{DR}$  will degenerate to the High Degree Adaptive (HDA) algorithm. The concept of HDA was proposed<sup>4</sup> as a better strategy and is slightly different from the original Degree centrality method. The degree of the remaining nodes in adaptive HDA is recomputed after each node removal.

To verify CI and  $CI_{DR}$  on the datasets, 2 implementations of the algorithm are utilized: CI\_HEAP<sup>40</sup> and ComplexCi<sup>41</sup>. CI\_HEAP was provided by the original paper written in the C language and generates the statistics of the Original Reinsertion method. ComplexCi is newly developed as a C++ implementation and produces the statistics of  $CI_{DR}$ . CI\_HEAP and ComplexCi share the same parameters as follows:

- The start points of the reinsertion in CI\_HEAP and ComplexCi are the same. Both start to reinsert a node when the size of the giant component collapses to 1% of the whole network.
- The finite fractions of nodes at each reinserted step in CI\_HEAP and ComplexCi are the same, and both methods reinsert 0.1% at each step.
- The intervals of the computing component in CI\_HEAP and ComplexCi are the same. To determine whether 1% of the giant component has been reached, CI\_HEAP and ComplexCi both need to compute the size of the giant component periodically. The interval parameter is 1%, which means that they will calculate the giant component after the CI algorithm removes 1% of the network nodes.

There are several differences between CI\_HEAP and ComplexCi when implementing their algorithms as follows.

- CI\_HEAP uses the Original Reinsertion method, and ComplexCi uses  $CI_{DR}$ .
- Compared with the initial proposed CI<sup>4</sup>, CI\_HEAP enhances the algorithm by utilizing the max-heap data structure<sup>22</sup> for very efficiently processing the CI values. The computational complexity of CI is  $O(N \log N)$  when removing nodes one by one, made possible through an appropriate data structure for processing CI. The ComplexCi application uses a red-black tree with the STL (Standard Template Library) container SET as a different data structure to store and update the CI values. In the field of C++ programming, the SET and MAP containers in STL are usually implemented as red-black trees, which are a type of self-balancing binary search tree. The average computational complexity of a red-black tree in searching, inserting and deleting nodes is  $O(\log N)$ . Although the red-black tree does not outperform the performance of deleting and updating, in contrast to max-heap, red-black tree is still able to achieve an overall computational complexity of  $O(N \log N)$ .
- As mentioned in algorithm 2, when the reinsertion is implemented in the experimental section, the top 0.1% of qualified nodes are added back at each step. For instance, if we have 2000 removed nodes, reinsertion will add back  $0.1\% * 2000 = 20$  nodes at each step until all nodes are once again in the network. Hence, we need to choose 20 nodes with the minimal value  $S_{min}$  out of the total of 2000 candidates at each reinsertion. Original Reinsertion implements a direct quick sort algorithm of  $O(N \log N)$  to sort all nodes and obtain the top nodes. In  $CI_{DR}$ , *Introselect algorithm*<sup>42</sup> is used to select the top N qualified nodes without a sort algorithm, therein simply being of  $O(N)$ . We do not need to know the order of the  $S_{min}$  array using full sort; we simply need to know the top N qualified nodes.

For Betweenness centrality<sup>19</sup>, Closeness centrality<sup>20</sup> and PageRank centrality<sup>21</sup>, a complex network python library GraphTools<sup>43</sup> is utilized to obtain the statistics. For Eigenvector centrality and Nonbacktracking centrality, the python tool NetworkX<sup>44</sup> is implemented to generate the statistics. To obtain the Nonbacktracking centrality of a network, if the leading eigenvector of its Nonbacktracking matrix B is computed directly according to the definition, the computational complexity will be high. In practice, a faster computation can be executed by utilizing the so-called Ihara (or Ihara-Bass) determinant formula<sup>31,45,46</sup>. It can be shown that the centralities on the Nonbacktracking matrix are equal to the first n elements of the leading eigenvector of the  $2N * 2N$  matrix in Formula 3:

$$M = \begin{pmatrix} A & I - D \\ I & 0 \end{pmatrix} \quad (3)$$

where A is the adjacency matrix, I is the identity matrix, and D is the diagonal matrix, with the degrees of the vertices along the diagonal<sup>24</sup>.

As mentioned above, the 8 competition datasets in the paper were obtained from the DataCastle Master Competition<sup>28</sup>, therein providing 4 real networks and 4 classical artificial networks. The 4 extra randomly generated graphs under the ER model are generated locally by the python utility NetworkX<sup>44</sup>.

For the calculation of the Robustness score, the code used in this paper is implemented from the DataCastle Master Competition and can be found at the official DataCastle website<sup>47</sup>.

## References

1. Lü, L. *et al.* Vital nodes identification in complex networks. *Phys. Reports* **650**, 1–63 (2016).
2. Rabade, R., Mishra, N. & Sharma, S. Survey of influential user identification techniques in online social networks. In *Recent advances in intelligent informatics*, 359–370 (Springer, 2014).
3. Lü, L., Zhang, Y.-C., Yeung, C. H. & Zhou, T. Leaders in social networks, the delicious case. *PLoS one* **6**, e21202 (2011).
4. Morone, F. & Makse, H. A. Influence maximization in complex networks through optimal percolation. *Nat.* **524**, 65 (2015).
5. Wang, J., Li, M., Wang, H. & Pan, Y. Identification of essential proteins based on edge clustering coefficient. *IEEE/ACM Transactions on Comput. Biol. Bioinforma.* **9**, 1070–1080 (2012).
6. Li, M., Zhang, H., Wang, J.-X. & Pan, Y. A new essential protein discovery method based on the integration of protein-protein interaction and gene expression data. *BMC systems biology* **6**, 15 (2012).
7. Luo, J. & Qi, Y. Identification of essential proteins based on a new combination of local interaction density and protein complexes. *PLoS one* **10**, e0131418 (2015).
8. Sarli, C. C. & Carpenter, C. R. Measuring academic productivity and changing definitions of scientific impact. *Mo. medicine* **111**, 399 (2014).
9. Garfield, E. The evolution of the science citation index. *Int. microbiology* **10**, 65 (2007).

10. Chen, P., Xie, H., Maslov, S. & Redner, S. Finding scientific gems with google's pagerank algorithm. *J. Informetrics* **1**, 8–15 (2007).
11. Kontogiorgos, E. J., Rustem, B. & Winker, P. *Computational methods in financial engineering* (Springer, 2008).
12. Gai, P. & Kapadia, S. Contagion in financial networks. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 466, 2401–2423 (The Royal Society, 2010).
13. Mistrulli, P. E. Assessing financial contagion in the interbank market: Maximum entropy versus observed interbank lending patterns. *J. Bank. & Finance* **35**, 1114–1127 (2011).
14. Feeley, T. H. & Barnett, G. A. Predicting employee turnover from communication networks. *Hum. Commun. Res.* **23**, 370–387 (1997).
15. Feeley, T. H., Hwang, J. & Barnett, G. A. Predicting employee turnover from friendship networks. *J. Appl. Commun. Res.* **36**, 56–73 (2008).
16. Pinzger, M., Nagappan, N. & Murphy, B. Can developer-module networks predict failures? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2–12 (ACM, 2008).
17. Bird, C., Nagappan, N., Gall, H., Murphy, B. & Devanbu, P. Putting it all together: Using socio-technical networks to predict failures. In *Software Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on*, 109–119 (IEEE, 2009).
18. Meneely, A., Williams, L., Snipes, W. & Osborne, J. Predicting failures with developer networks and social network analysis. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 13–23 (ACM, 2008).
19. Freeman, L. C. A set of measures of centrality based on betweenness. *Sociom.* 35–41 (1977).
20. Bavelas, A. Communication patterns in task-oriented groups. *The J. Acoust. Soc. Am.* **22**, 725–730 (1950).
21. Page, L., Brin, S., Motwani, R. & Winograd, T. The pagerank citation ranking: Bringing order to the web. *Tech. Rep., Stanford InfoLab* (1999).
22. Morone, F., Min, B., Bo, L., Mari, R. & Makse, H. A. Collective influence algorithm to find influencers via optimal percolation in massively large social media. *Sci. reports* **6** (2016).
23. Freeman, L. C. Centrality in social networks conceptual clarification. *Soc. networks* **1**, 215–239 (1978).
24. Martin, T., Zhang, X. & Newman, M. Localization and centrality in networks. *Phys. Rev. E* **90**, 052808 (2014).
25. Teng, X., Pei, S., Morone, F. & Makse, H. A. Collective influence of multiple spreaders evaluated by tracing real information flow in large-scale social networks. *Sci. reports* **6** (2016).
26. Schneider, C. M., Moreira, A. A., Andrade, J. S., Havlin, S. & Herrmann, H. J. Mitigation of malicious attacks on networks. *Proc. Nat. Acad. Sci.* **108**, 3838–3841 (2011).
27. Tarjan, R. E. Efficiency of a good but not linear set union algorithm. *J. ACM (JACM)* **22**, 215–225 (1975).
28. DataCastle. Datacastle master competition background, [http://www.dcjingsai.com/common/cmpt/%E5%A4%A7%E5%B8%88%E8%B5%9B\\_%E7%AB%9E%E8%B5%9B%E4%BF%A1%E6%81%AF.html](http://www.dcjingsai.com/common/cmpt/%E5%A4%A7%E5%B8%88%E8%B5%9B_%E7%AB%9E%E8%B5%9B%E4%BF%A1%E6%81%AF.html) (2017).
29. Tarjan, R. E. & Van Leeuwen, J. Worst-case analysis of set union algorithms. *J. ACM (JACM)* **31**, 245–281 (1984).
30. Tarjan, R. E. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. computer system sciences* **18**, 110–127 (1979).
31. Hashimoto, K. -I. Zeta functions of finite graphs and representations of p-adic groups. In *Automorphic forms and geometry of arithmetic varieties*, 211–280 (Elsevier, 1989).
32. Krzakala, F. et al. Spectral redemption in clustering sparse networks. *Proc. Nat. Acad. Sci.* **110**, 20935–20940 (2013).
33. Newman, M. Spectral community detection in sparse networks. *arXiv preprint arXiv:1308.6494* (2013).
34. Erdos, P. On random graphs. *Publ. mathematicae* **6**, 290–297 (1959).
35. Tianyu, W., Zhang, J. & Wandelt, S. Exploiting global information in complex network repair processes. *Chin. J. Aeronaut.* **30**, 1086–1100 (2017).
36. Lee, Y. H. & Sohn, I. Reconstructing damaged complex networks based on neural networks. *Symmetry* **9**, 310 (2017).
37. Hu, F., Yeung, C. H., Yang, S., Wang, W. & Zeng, A. Recovery of infrastructure networks after localised attacks. *Sci. reports* **6**, 24522 (2016).
38. Farr, R. S., Harer, J. L. & Fink, T. M. Easily repairable networks: Reconnecting nodes after damage. *Phys. review letters* **113**, 138701 (2014).
39. Sun, W. & Zeng, A. Target recovery in complex networks. *The Eur. Phys. J. B* **90**, 10 (2017).
40. Makse, H. A. C code implementation of collective influence algorithm, [http://www-levich.engr.ccnyc.cuny.edu/hernanlab/uploads/CI\\_HEAPc](http://www-levich.engr.ccnyc.cuny.edu/hernanlab/uploads/CI_HEAPc) (2016).
41. Zhu, F. zhftk/complexci, <https://github.com/zhftk/ComplexCi> (2017).
42. Wikipedia. Introselect–Wikipedia, the free encyclopedia, <http://en.wikipedia.org/w/index.php?title=Introselect&oldid=848338425>, [Online; accessed 29-July-2018] (2018).
43. Peixoto, T. P. The graph-tool python library. *figshare*, <http://figshare.com/articles/graphtool/1164194>, <https://doi.org/10.6084/m9.figshare.1164194> (2014).
44. Hagberg, A., Swart, P. & S Chult, D. Exploring network structure, dynamics, and function using networkx. Tech. Rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (2008).
45. Bass, H. The ihara-selberg zeta function of a tree lattice. *Int. J. Math.* **3**, 717–797 (1992).
46. Angel, O., Friedman, J. & Hoory, S. The non-backtracking spectrum of the universal cover of a graph. *Transactions Am. Math. Soc.* **367**, 4287–4318 (2015).
47. DataCastle. Source code of scoring criteria in robustness calculation, [http://share.pkbigdata.com/ID.4407/Master\\_algorithm](http://share.pkbigdata.com/ID.4407/Master_algorithm) (2017).

## Acknowledgements

The author thanks Hernan Makse for discussing the idea in the mail and Shanwen Zhu for helping improve the quality of the paper. The author also thanks American Journal Experts (AJE) for English language editing. This manuscript was edited for English language by American Journal Experts (AJE).

## Author Contributions

F.Z. designed the research, analyzed the data, prepared the figures and wrote the main manuscript text.

## Additional Information

**Competing Interests:** I declare that the author has no financial competing interests. For the non-financial interests, F.Z. is a participant in the DataCastle Master Competition and write the manuscript from the research in the competition.

**Publisher's note:** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

© The Author(s) 2018