Scalable heuristic algorithm for identifying critical nodes in networks

Dalaijargal Purevsuren[a,*], Gantulga Gombojav[a,**], Gang Cui[b]

[a] *Department of Information and Computer Sciences, School of Engineering and Applied Sciences, National University of Mongolia, Ulaanbaatar 14201, Mongolia*

[b] *School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China*

Version 1
November 21, 2022

Abstract

This paper presents two heuristic algorithms for the distance-based critical node problem (DCNP) that finds k nodes whose removal minimizes the pairwise connection within D hops in the remaining network. The structural properties of the complex networks have not yet been extensively addressed in the literature. Specifically, the community structure of complex networks needs to be considered more in the literature. In this study, we propose a network destruction strategy by extracting the community structure, one of the main structural properties of complex networks. The proposed strategy enables us to develop a scalable greedy algorithm for the DCNP. In addition, a simple evolutionary algorithm with a novel gene representation obtained by the proposed destruction strategy is investigated for the problem. We tested our algorithms on well-known complex networks and compared them with two recently proposed other algorithms for performance evaluation. The proposed greedy algorithm works orders of magnitude faster than other methods and provides comparable solution quality. Our evolutionary algorithm finds the optimal values on most tests.

Keywords:
critical node problem, community structure, combinatorial optimization, scalable algorithms, evolutionary algorithms, collective attacker, betweenness centrality

[*] e-mail: dalaijargal@gmail.com
[**] e-mail: gantulgag@seas.num.edu.mn

1. INTRODUCTION

Nodes in networks have different roles. Identifying critical (key) nodes (players) has practical applications in many areas, including managing telecommunication network [1, 2] planning effective vaccination [1], assessing the vulnerability of transportation networks during disaster [3, 1], understanding the resilience of river networks under potential drainage [4], detecting vulnerable nodes of power networks [5], dismantling terrorist communication network [6], distracting illicit trade network [7, 8], analyzing protein-protein networks [9].

The network consists of the set of vertices V with $V = n$ and the set of edges E with $E = m$. Given a network (graph) $G(V, E)$, distance (the number of hops) D and a constraint integer k on the number of nodes to remove, the distance-based critical node problem (DCNP) [10, 11, 12, 13] identifies a set of k vertices whose removal minimize the pair-wise connectivity within D hops in the remaining network. The mathematical formulation can be given as an optimization problem with Equation (1) and (2), where $f(S)$ is the objective function, S is the set of nodes to be deleted from G.

$$f(S) = \min\{|\text{paths within } D \text{ hops in in } G[V \setminus S]|\} \tag{1}$$

$$\text{s. t. : } |S| \leq k \tag{2}$$

The critical node problem (CNP), which is first introduced in [1, 14], has several variants that depend on their connectivity measure, i.e. total pairwise connectivity (the classic CNP [14, 15, 16], distance-based critical node problem (DCNP [10, 11, 12, 13]), and the largest remaining connected component (the CC-CNP [17, 18, 19]). The classic CNP and its variants are NP-hard on a general graph [20, 14, 21]. For more information about the variants and methods, interested readers can refer to [22]. Note that we denote the variants of critical node problems by the CNPs.

The most common property of complex networks used in CNPs is network centralities. Network centralities, including degree, betweenness, closeness, eigenvalue, and Katz, are studied in approaches for the CNPs [15, 16, 12, 13]. In [15, 16, 12], their algorithm uses betweenness centrality (BC) to rank nodes to find what to remove. The authors [16] point to the effectiveness of BC for driving search, but the computational expensiveness of BC makes BC-based approaches less competitive than others. The constructive approach in [16] is modified in [13] by selecting double budget nodes with the highest BC for removal. Conceptually, the approaches [15, 16, 12, 13] use BC as a node centrality measurement, not a community detector. Note that BC can work as centrality measurement as in [23] but also be used in community detection [24, 25].

The betweenness centrality (BC) [23] computes a node centrality based on how many times the node lies on the paths between all other pairs of nodes. BC has a version that considers the distance between nodes in BC computation [26], and we call it d-BC. The d-BC computes BC from only paths with at most D lengths.

Networks are organized with communities [24, 27], in which sub-networks have more links between themselves and fewer links with others. The idea of the Girvan-Newman (GN) algorithm [24], the most known approach in community detection, finds and removes inter-community edges iteratively. Therefore, the communities remain in the end.

Although many approaches study network centralities for CNPs, the community structure, which has been studied extensively in the last two decades [27], has not yet been investigated in the literature of the CNPs, except addressing the application of CNPs for the community detection [28, 29, 17]. As far as we know, this work is the first attempt to integrate community structure with the approach for solving the DCNP. In addition, d-BC is used to capture distance information instead of ordinary BC.

We extend the idea of the GN algorithm by removing multiple (e.g., $\sqrt{n}$) nodes simultaneously (instead of single edges at each iteration) and re-inserting some of them. The edges and nodes in inter-communities have high betweenness centralities [24, 30]. Neighbors of the node with the highest BC have a high chance of having high BC when they are also located in the inter-community region. Therefore, the top k nodes with the highest BC may locate in the same region. After removal and re-insertion operations, we call the resulting nodes the collective attacker nodes. The collection of k nodes with the highest BC is selected for removal.

The proposed strategy (collective attacker) enables us to develop a scalable greedy heuristic for the DCNP. The proposed algorithm is tested on well-known real-world networks and compared with two recently proposed algorithms for performance evaluation. Our greedy algorithm works much faster than other methods and provides comparable

solution quality. In addition, a simple evolutionary algorithm with a novel gene representation obtained by the proposed destruction strategy is investigated for the problem. Our evolutionary algorithm finds the optimal values on most networks used in the experiments within a shorter time than others.

The paper is organized as follows. Section two presents the proposed destruction strategy, greedy heuristic, and evolutionary algorithm. Section three presents the results of the computational experiments. The conclusions and possible future extensions are given in section four.

2. TWO HEURISTICS FOR THE DISTANCE-BASED CRITICAL NODE PROBLEM

This section presents two heuristics for the distance-based critical node problem (DCNP). At first, a new destruction strategy is proposed, and then a scalable greedy heuristic is designed for the DCNP. After that, the greedy heuristic is extended with a simple evolutionary scheme, similar to $(\mu + \lambda)$-EA.

2.1. Collective attacker

Some nodes in a network have different roles and impacts. Even though a node has a small impact on the destruction of the network as an individual, this node may have a high impact with other strategically positioned nodes (e.g. neighbors of neighbors) as a part of a crew. The impact of a collection of nodes can be totally different from a simple aggregation of individual impacts. The collective impact of a collection of nodes needs to be measured as a unit. One way to measure the collective impact of some nodes is to examine the structural differences in the remaining networks after removing or deactivating those nodes.

Definition 1. A collection of nodes with a high collective impact on destruction is called the collective attacker in complex networks.

The detection of the collective attacker consists of two phases: (1) detection phase, which detects a collection of nodes from inter-communities (2) re-insertion phase, which re-evaluates the detected nodes from a different aspect. The detection phase uses the betweenness centrality (BC) [23], which computes a node centrality based on how many times the node lies on the paths between all other pairs of nodes. An illustration of BC on a sample network is given in Figure 1 (B), where nodes with high BC are in bold. The nodes A, B, C, D and E get higher BC scores. Assume that A and C nodes are removed in the first attempt. In this case, the network is divided into two components. If we execute BC on the remaining network, two inter-community nodes with 13 and 8 scores (see Figure 1 (C)) are highlighted. Their removal divides the network into three components with two sequences of CA removal.

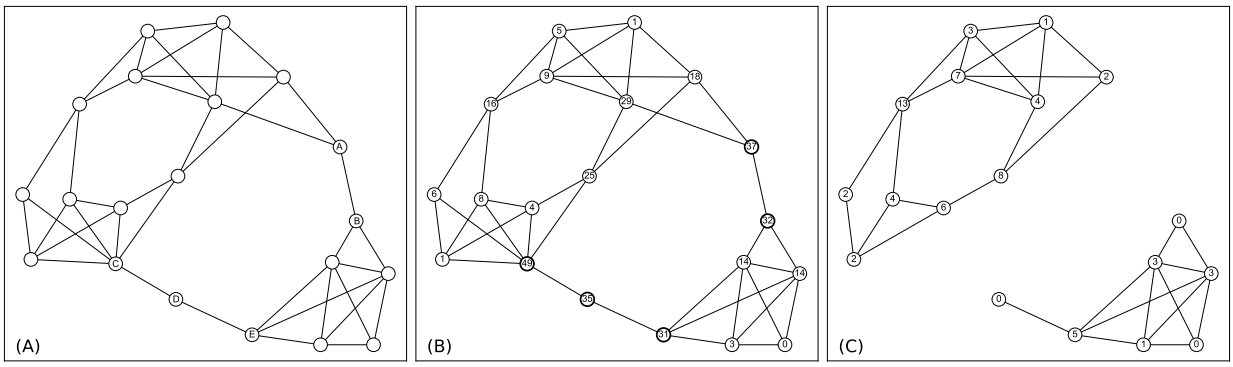

Figure 1. Sample network. (A) a network has two communities; the five labeled nodes are on inter-community and boundary regions. (B) betweenness centrality (BC) is assigned to each node; five nodes with the highest centrality are in bold. (C) BC on the remaining network after nodes are removed

Definition 2. The collection of nodes with the highest betweenness centrality is called the critical region in complex networks.

Although removing all the nodes in the critical region dissects the network well, it may include unnecessary nodes. For example, assume that the nodes A, B, C, D and E are the critical region of the network in Figure 1 (A). It is unnecessary to remove all five nodes; a pair of them can divide the network into two groups. Therefore, a re-insertion phase is applied to avoid unnecessary removal. In the re-insertion phase, the selected k nodes are re-evaluated one by one. If there is not much structural difference in the network between with and without the node, re-insert it into the remaining network. This phase helps minimize the total number of nodes removed without hurting the collective impact of the critical region obtained in the detection phase. The nodes after re-insertion are the collective attacker.

Algorithm 1 CA-extraction algorithm

CA-EXTRACT(G, B)
1: Calculate approximate d-BC of G
2: $S = \{v_i : \sqrt{B}$ nodes with the highest d-BC$\}$
3: $G = G([V \backslash S])$
4: **return** G, S

2.2. Scalable greedy heuristic for the DCNP

This sub-section presents a scalable greedy heuristic using collective attacker (CA). Iterative call of CA-extraction is investigated here. Removing collective attackers does not guarantee network separation but makes the network vulnerable to attack through the critical region. These procedures are given in Algorithm 2 and 3 in detail.

First, our re-insertion algorithm ranks nodes from low to high impact in destruction according to the following H (Equation 3) function, which measures the destruction quantity of a node u to be restored from the graph S, where C is the set of connected components in $G[V \backslash (S \backslash u)]$ graph, and h is the size of the components in C.

$$H(G, S, u) = \sum_{h \in C} \frac{h(h-1)}{2} \tag{3}$$

The lesser value of H means a better set. That means the critical region S cuts the graph G into smaller sizes of connected components since it is computed by squaring the size of connected components in the remaining graph. To evaluate node u's impact, after restoring it $(S \backslash u)$, the higher value of H implies that node u has a high impact member of S.

Algorithm 2 CA-greedy algorithm

CA-GREEDY(G, B)
1: $S = \emptyset$
2: **while** $|S| < B$
3: $\quad [G, S'] = CA - extract(G, B)$
4: $\quad S = S \cup S'$
5: re-insert(G, S, B)
6: **return** S

Algorithm 3 Re-insertion algorithm

RE-INSERT(G, S, B)
1: $L = |S| - B$
2: Sort S by $H(G, S, u)$, where $u \in S$
3: Restore first $2L$ nodes in S
4: Calculate approximate d-BC on G
5: Delete top L nodes with the highest d-BC from G, and insert them into S
6: **return** S

2.3. Evolutionary algorithm for the DCNP

In this sub-section, we present a simple evolutionary algorithm for the DCNP (called CA-evo). In CA-greedy, the solution is constructed with the first m CAs with sequential removal of CA-removal algorithm ($S = CA_1 \cup CA_2 \cup \cdots \cup CA_m$, called elite-m). There is no guarantee that the first CAs can find the best solution for the DCNP. Let us consider a case that one CA misses from the elite-m. In this case, we need one more CA to make elite-m complete. The new CA added is totally different from the missing one. The elite-m with new CA can be an interesting solution to further investigation. In general, we can consider many other cases that many CAs miss from the elite-m and new CAs are added to make it complete. Our CA-evo is simple, similar to (2 + 2)−GA [31]. It has only cross-over and repair operator, not explicit mutation and diversity mechanism. We uses determistic ($\mu + \lambda$) selection strategy. It has several building blocks: representation, initialization, recombination, repair operator and termination condition. The algorithm 4 and 5 describes CA-evo in detail. We set population size (N) by 3.

The representation of the CA-evo is the main contribution of this algorithm. CA-evo takes a CA as a gene in the chromosome of an individual. An individual has a pre-defined number of genes. The size of a gene is $\sqrt{B}$. We denote the chromosome length by CL, which is computed by the following formula, $CL = \max(c \cdot \sqrt{b} \cdot \log_{10}(n), 5)$, where c is a constant.

Initialization is an important part of the good performance of the CA-evo, according to the preliminary experiments. At first, this step collects CAs until the network becomes small enough. We call these collected CAs the elite genes. Note that elite-m collects CAs until the total number of nodes in elite-m becomes not less than the constraint k. Individuals are randomly generated by selecting genes from the elite genes. Note that the number of elite genes may be less than the length of chromosomes. Another word, many replications of a gene may occur in an individual.

Recombination (new-generation) generates new offspring from the randomly chosen two parents of the current population. Specifically, it selects CL genes from the union of genes of its parents at random. If the new offspring is infeasible, it calls the repair operator. Recombination generates N offspring. Note that each individual has the same chance to be selected for mating. The best N individuals (ordered by OFV) from the union pool of parents and children are survived for the next generation.

The repair operator (RO) also uses the CA-extraction algorithm. In the repair step, CA-extraction is called on the remaining network after removing all nodes in the offspring. This extraction process continues until the offspring becomes feasible.

The CA-evo checks two termination conditions, total running time (T_{max}) and the number of generations with no improvement on the best individual ($idle_{max}$), which helps the algorithm avoid unnecessary running.

Algorithm 4 CA-evo algorithm

CA-EVO(G, T_{max}, B, D)
1: $N = 3$
2: $P = \text{Init}(G, N)$
3: $idle = 0, idle_{max} = \max(100, 10 \cdot B)$
4: **while** $t < T_{max}$ and $idle < idle_{max}$
5: $P' = \text{new-generation}(P, N)$
6: order $P = P' \cup P$ with respect to objective function
7: $P = \{p_i : p_i \in P, i < N\}$
8: **return** nodes in P_0

Algorithm 5 New-generation

NEW-GENERATION(G, N)
1: $P = \text{cross-over}(P)$
2: $P = \text{repair}(G, P)$
3: **return** P

3. COMPUTATIONAL EXPERIMENTS

In this section, the proposed algorithms are computationally analyzed and tested for performance evaluation. We compared our algorithms against two recent works on DCNP which are [11] and [13]. The algorithm presented in [11] is based on memetic search and we will call this algorithm "memetic algorithm" (MA). The algorithm presented in [13] is based on greedy 3 and we will call this algorithm "greedy 3" (GR 3).

3.1. Experimental setup

We used the Networkit graph library configured to use a single thread and developed our code in Python 3 [32]. Our experiments are run on a normal desktop computer with Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz with 8GB ram. We use a time limit of $T_{max} = 3,600$ seconds for all execution.

3.1.1. Dataset

We will call the datasets used in [11] and [13] as benchmark 1 and 2 respectively. We used the same real world datasets used in these works. These datasets are downloaded from Pajek dataset [33], 10^{th} DICMACS Implementation Challenge [34] and UCINET software datasets [35]. We remapped the nodes indices and formatted these datasets making it ready to use in our algorithm because the datasets downloaded from different sources were in different formats. Anyone who wishes to try our algorithm can download the dataset from github repository. Quantitative information such as number of vertices and number of edges and structural information such as diameter of the graph and number of paths whose length is within k distance are shown in Tables 2 and 1.

The networks used in benchmark 1 are small as shown in Table 1. The biggest network is *netscience* collaboration network containing 379 vertices.

Table 1. Benchmark 1 dataset

Network	V	E	E^3	Diam
Hi-tech	33	91	466	5
karate	34	78	480	5
mexican	35	117	583	4
Sawmill	36	62	397	8
chesapeake	39	170	741	3
dolphins	62	159	1,107	8
Lesmiserable	77	254	2,500	5
Santafe	118	200	2,273	12
Sanjuansur	75	155	1,352	7
Attiro	59	128	1,164	8
LindenStrasse	232	303	3,251	13
SmallWorld	233	994	25,721	4
netscience	379	914	9,523	17
USAir97	332	2,126	46,573	6

Networks used in benchmark 2 are a mixture of medium and small sized graphs – the smallest among them is *karate* social network with 34 vertices and the biggest among them is *cond-mat* collaboration network with $16,726$ vertices.

The budget sizes of some graphs in benchmark 1 become 1 since the datasets used in their work are small. This makes a good heuristic algorithm show a bad performance. It is hard to make a reasonable judgement when the budget is too small. Moreover, the budget size in benchmark 2 is only 5 and 10 even though the dataset used in benchmark 2 is mixed. His objective was to develop an optimal algorithm and the complexity grows exponentially when the budget increases. Therefore, these datasets and budget sizes don't show a scalable nature of ca-greedy. We cherry picked the following big datasets with 5% and 10% budget size and run our method to present some evaluation data for future research works in this field.

3.2. Algorithm configuration

CA-greedy doesn't require any algorithmic parameter. It only takes b budget size and k distance limit. We tried to make the parameters of CA-evo minimum. Nevertheless, it requires some parameter since it's an evolutionary algorithm. Our evolutionary algorithm depends on the parameters shown in Table 6. Chromosome length (CL) defines the number of genes in a single entity of the population. All the nodes contained in one's genes add up to k and give

Table 2. Benchmark 2 dataset

Network	V	E	E^3	Diam
karate	34	78	480	5
dolphins	62	159	1,107	8
lesmis	77	254	2,500	5
LindenStrasse	232	303	3,251	13
polbooks	105	441	3,510	7
adjnoun	112	425	5,634	5
football	115	613	6,247	4
netscience	1,589	2,742	13,087	17
jazz	198	2,742	18,461	6
SmallWorld	233	994	25,721	4
Erdos971	429	1,312	34,086	11
S.Cerevisae	1,458	1,948	39,091	19
USAir97	332	2,126	46,573	6
power	4,941	6,594	53,125	46
H.Pylori	706	1,392	62,028	9
Harvard500	500	2,043	83,993	6
homer	561	1,628	91,537	9
celegans_metabolic	453	2,025	91,531	7
email	1,133	5,451	289,259	8
hep-th	8,361	15,751	376,431	19
PGPgiantcompo	10,680	24,316	1,145,492	24
cond-mat	16,726	47,594	1,761,969	18

us a solution for DCNP. As we mentioned before, this is a variable length and it can be changed by the repair operator to meet the problem requirement. We used a very small population size (N) which is 3 and this makes our algorithm run fast and converge rapidly. This is needed since our algorithm being based on betweenness value already gives a quite good result and we just needed to search around to improve the result even further. The parameter maximum number of idle iteration ($idle_{max}$) sets the number of iteration that the algorithm is allowed to run without improving its result. This parameter affects the quality of the solution and the running time directly because, the algorithm will get more time to search when this parameter is set to a bigger value. Gene size (GS) defines the number of nodes to delete in a single iteration of CA-greedy. We use the output of CA-greedy as genes in the evolutionary algorithm. This number needs to be dependent on the budget size. Parameter tuning for a heuristic algorithm is often a tedious and time consuming work. In order to fine tune the algorithm, we tried a range of values for a specific parameter while other parameters are fixed at the values that are known to perform well. The networks with more than a hundred nodes are considered to make the changes more obvious since small networks are trivial.

First, we studied the suitable values for CL in Table 3. This parameter has a direct correlation with the gene size which in turn depends on the budget size b. Therefore, we tried different values for the coefficient c_1 in Equation 4 for CL value.

$$CL = \max(c_1 \cdot \sqrt{b} \cdot \log_{10} n, 5) \tag{4}$$

In Table 3, the column c_1 is the ceofficent of CL in the equation, the column CL is the actual length of the chromosome and the columns OFV and time are the averages of the objective function and time over 30 runs respectively. Other parameters are set as $I = 2000, N = 3$ and $GS = \sqrt{b}$ while we change c_1 parameter. The running time decreases for some time when c_1 increases and it increases again at certain point because, the algorithm converges fast and hits an idle iteration limit when the population contains more information until this information becomes too much and slows down the algorithm. The same behavior is observed for the objective function value. When an individual has more genes which contains critical nodes survived long enough, the quality of the solution increases. But it starts to decline when the length becomes too big because, it gives more pressure to the repair operator and it being based on greedy nature can't sort out the critical nodes well. The suitable values for c_1 are 2 and 3 as we can observe from Table 3. We will set $c_1 = 2$ for further comparisons as it shows a slightly better solution quality.

Secondly, we studied the effect of gene size (GS) on the solution quality and running time. GS is the number of nodes to be deleted in a single iteration of CA-greedy and directly depends on b. We used Equation 5 for GS size and iterated over several values for c_2 coefficient in order to analyze this parameter.

$$GS = c_2 \cdot \sqrt{b} \tag{5}$$

Table 4 shows that the running time increases when gs is small and it decreases as it grows bigger. It requires more iterations in the initialization step of the algorithm and longer chromosome when the number of the nodes contained

Table 3. The performance when *CL* parameter changes according to Equation 4. OFV and time are the averages of the objective function and running time over 30 runs respectively. c_1 is the coefficient in the equation. $N = 3, GS = \sqrt{b}$ and $I = 2000$ in this experiment

Networks	b	$c_1 = 0.5$			$c_1 = 1$			$c_1 = 2$		
		CL	OFV	time	CL	OFV	time	CL	OFV	time
Santafe	11	5	116.3	3.09	6	116.1	2.78	8	116.1	2.6
LindenStrasse	23	5	446.5	13.31	11	436.1	3.99	22	435.3	8.27
SmallWorld	23	5	1,775.7	83.38	11	1,705.7	47.95	22	1,702.6	33.27
netscience	37	7	944.3	28.16	15	916.9	26.54	31	907.5	25.38
USAir97	33	7	3,507.4	190.29	14	3,319.0	116.91	28	3,296.8	94.94
		$c_1 = 3$			$c_1 = 4$			$c_1 = 5$		
Santafe	11	20	116.4	2.59	27	116.1	2.51	34	116.5	2.71
LindenStrasse	23	34	433.8	9.41	45	437.5	8.1	56	436.5	7.98
SmallWorld	23	34	1,702.3	38.9	45	1,712.7	29.24	56	1,712.7	26.88
netscience	37	47	907.8	24.06	62	912.3	25.33	78	915.3	21.76
USAir97	33	43	3,323.8	67.84	57	3,341.2	71.41	72	3,345.8	65.7

in a gene is small. Thus, it slows down the algorithm. Small GS shows a better performance in terms of quality since it produces different varieties of genes making the algorithm more precise. We can see that the sweet spot for GS is 1 in this experiment.

Table 4. Performance when GS is changed according to Equation 5. OFV and time are the averages of the objective function and running time over 30 runs respectively. c_2 is the coefficient in the equation. $N = 3, I = 5000$ and $c = 2$ in this experiment.

		$c_2 = 0.5$			$c_2 = 1$			$c_2 = 1.5$			$c_2 = 2$		
	b	gs	OFV	time	gs	OFV	time	gs	OFV	time	gs	OFV	time
Santafe	11	2	116.2	7.42	3	116.0	7.66	4	116.4	7.11	6	116.5	8.6
LindenStrasse	23	2	435.8	23.58	5	434.3	17.49	7	436.6	24.04	8	436.2	23.71
SmallWorld	23	2	1706.5	84.86	5	1700.6	70.8	14	1,699.0	90.47	17	1,729.3	76.57
netscience	37	3	911.1	53.11	6	904.4	45.1	11	905.1	66.39	13	908.4	54.99
USAir97	33	3	3295.0	202.49	6	3307	181.27	21	3,329.0	180.1	27	3,338.8	145.28
Total			6464.6	371.46		6462.2	322.32		6,486.0	368.11		6,529.3	309.15

Lastly, we experimented $idle_{max}$ parameter on four different values in Table 5. This parameter gives CA-evo algorithm more time to search for further improvement. It halts the algorithm if the algorithm runs $idle_{max}$ iteration without any improvements.

Table 5. The parameters are $N = 3$, $b = 0.1n$, $c_1 = 2$ and $c_2 = 1$ in this experiment.

Networks	$idle_{max} = 100$				$idle_{max} = 1000$				$idle_{max} = 5000$				$idle_{max} = 10000$			
	avg	btime	time	ratio	avg	btime	time	ratio	avg	btime	time	ratio	avg	btime	time	ratio
Santafe	116.3	0.05	0.17	0.29	116	0.04	0.99	0.04	116	0.06	7.66	0.01	116.0	0.33	13.18	0.03
LindenStrasse	437.7	0.21	0.52	0.40	436	0.75	3.05	0.25	434	2.74	17.49	0.16	433.1	4.55	34.66	0.13
SmallWorld	1,711.8	0.98	2.25	0.44	1,707	2.19	11.46	0.19	1,701	7.84	70.8	0.11	1,697.3	7.91	133.62	0.06
netscience	922.1	0.77	1.44	0.53	911	2.04	6.67	0.31	904	12.14	45.1	0.27	902.9	18.48	82.61	0.22
USAir97	3,379.1	2.69	5.54	0.49	3,341	13.62	33.73	0.40	3,307	37.87	181.27	0.21	3,294.9	68.16	360.73	0.19

We ran our algorithm on benchmark 1 and benchmark 2 with the parameter configuration shown in Table 6.

Table 6. Parameter configuration for CA-evo

Parameter	**Algorithm configuration**
Length of chromosome (CL)	$\max(2 \cdot \sqrt{b} \cdot \log_{10} n, 5)$
Population size (N)	3
Number of the iterations with no improvement ($idle_{max}$)	100 or 5000
Gene size (GS)	$\sqrt{b}$

3.3. Comparison with other methods

CA-evo and CA-greedy are compared against MA on benchmark 1 dataset with the budges sizes $b = 0.05n$ and $b = 0.1n$ in Tables 7 and 8 respectively.

CA-evo and CA-greedy are compared against GR3 on benchmark 2 dataset with the budget size $b = 5$ and $b = 10$ in Tables 9 and 10 respectively.

Table 7. Comparison with MA ($b = 0.05n$, $k = 3$)

		$b = 0.05n,\ c_1 = 2,\ c_2 = 1$											
Networks	opt	MA			CA-evo-i100			CA-evo-i5000			CA-greedy		
		obj	avg	time	obj	avg	time	obj	avg	time	obj	avg	time
Hi-tech	397	397	397	0.2	412	412.9	0.11	412	413.2	5.03	427	427.0	0.06
karate	324	324	324	0.2	324	327.6	0.09	324	324.0	4.16	324	324.0	0.03
mexican	527	527	527	0.3	527	527.0	0.12	527	527.0	5.78	527	527.0	0.05
Sawmill	215	215	215	0.2	215	215.0	0.05	215	215.0	2.69	215	215.0	0.03
chesapeake	696	696	696	0.3	696	696.0	0.16	696	696.0	7.51	696	696.0	0.05
dolphins	820	820	820	1.3	830	840.2	0.27	830	837.9	10.95	828	828.2	0.14
Lesmiserable	930	930	930	1.9	930	948.7	0.3	930	943.0	12.9	952	952.0	0.13
Santafe	305	305	305	1.2	305	305.0	0.15	305	305.0	7.14	312	312.0	0.07
Sanjuansur	803	803	803	0.9	803	803.1	0.23	803	803.0	10.37	803	803.0	0.14
Attiro	743	743	743	0.5	743	743.0	0.21	743	743.0	9.94	743	743.0	0.11
LindenStrasse	1054	1054	1057.8	5.9	1,054	1,059.5	0.61	1,054	1,054.4	25.86	1,175	1,178.2	0.18
SmallWorld	4629	4629	4660.5	51.4	4,629	4,651.8	3.34	4,629	4,632.5	112.78	4,825	4,828.9	0.49
netscience	2102	2102	2102	52.9	2,102	2,126.9	1.86	2,102	2,110.0	50.57	2,429	2,429.0	0.28
USAir97	10623	10,623	10,697.20	269.7	10,623	10,770.2	10.29	10,623	10,644.3	367.58	10,733	10,733.0	1.39

Table 8. Comparison with MA. ($b = 0.1n$, $k = 3$)

		$b = 0.1n,\ c_1 = 2,\ c_2 = 1$											
Networks	opt	MA			CA-evo-i100			CA-evo-i5000			CA-greedy		
		obj	avg	time	obj	avg	time	obj	avg	time	obj	avg	time
Hi-tech	293	293	294.8	0.5	293	297.0	0.11	293	293.7	4.62	302	302.0	0.14
karate	147	147	150.9	0.4	147	148.3	0.06	147	147.0	2.7	147	147.0	0.04
mexican	358	358	358	0.6	358	358.0	0.11	358	358.0	5.71	358	358.0	0.11
Sawmill	135	135	135	0.2	135	135.4	0.05	135	135.0	2.66	142	142.0	0.05
chesapeake	512	512	515.2	1.1	512	512.0	0.17	512	512.0	9.47	512	512.0	0.15
dolphins	583	583	591.7	2.5	583	584.0	0.27	583	583.1	12.4	654	654.4	0.14
Lesmiserable	323	323	323	2.9	323	323.0	0.19	323	323.0	11.1	323	323.0	0.13
Santafe	116	116	116	2.8	116	116.3	0.17	116	116.0	7.66	119	119.3	0.09
Sanjuansur	457	457	457.2	2	457	457.8	0.22	457	457.5	8.75	490	493.5	0.16
Attiro	444	444	450.4	0.7	444	444.2	0.2	444	444.1	7.41	465	465.0	0.12
LindenStrasse	429	429	431.7	12.7	430	437.7	0.52	429	434.3	17.49	475	480.9	0.28
SmallWorld	1,694	1,694	1,694	58.6	1,694	1,711.8	2.25	1,694	1,700.6	70.8	1,841	1,855.8	0.59
netscience	897	897	901	112.3	897	922.1	1.44	897	904.4	45.1	995	1,004.3	0.28
USAir97	3,100	3,100	3,219.1	423	3,168	3,379.1	5.54	3,123	3,307	181.27	3,788	3,804.3	1.45

Table 9. Comparison with greedy 3. ($b = 5$, $k = 3$)

		$b = 5$							
Networks	opt	GR3		CA-evo			CA-greedy		
		OFV	time	OFV	avg	time	OFV	avg	time
karate	41	41	0	41	41.9	0.05	41	41.0	0.04
dolphins	662	678	0.01	662	663.0	0.27	693	693.7	0.15
lesmis	517	535	0.01	517	520.6	0.27	535	535.0	0.14
LindenStrasse	1,810	1,815	0.09	1,810	1,810.0	0.66	1,837	1,850.4	0.19
polbooks	2,555	2,673	0.04	2,573	2,601.9	1.13	2,752	2,752.9	0.36
adjnoun	3,719	3,719	0.06	3,719	3,719.0	1.44	3,805	3,805.0	0.45
football	5,362	5,262	0.06	5,362	5,378.5	3.07	5,387	5,412.2	0.45
netscience	8,390	8,898	0.31	8,390	8,392.2	9.08	8,785	8,803.5	0.32
jazz	16,136	16,185	0.22	16,136	16,166.2	21.14	16,491	16,514.7	2.36
SmallWorld	6,964	6,964	0.14	6,964	6,964.0	5.94	6,964	6,964.0	0.59
Erdos971	25,737	25,737	0.56	25,737	25,823.6	21.30	26,136	26,168.3	1.61
S.Cerevisae	25,190	25,190	2.84	25,190	25,200.9	18.11	25,190	25,335.7	0.65
USAir97	29,486	29,486	0.3	29,486	29,559.2	26.74	30,021	30,065.9	2.86
power	50,410	52,456	39.36	50,410	50,411.6	82.51	51,330	51,479.9	1.44
H.Pylori	37,626	37,626	0.93	37,626	37,626.0	24.80	37,626	37,788.0	2.53
Harvard500	16,448	19,241	0.41	16,448	16,743.7	17.98	19,241	19,241.0	1.85
homer	45,828	45,828	0.56	45,838	45,838.0	31.29	46,807	46,807.0	3.10
celegans_metabolic	44,967	44,967	0.63	44,967	44,967.0	33.46	46,488	46,488.0	3.62
email	263,409	263,409	3.36	263,409	263,807.3	253.83	264,325	264,836.4	9.27
hep-th	345,320	345,320	77.05	345,320	345,320.0	451.96	351,233	351,233.0	4.85
PGPgiantcompo	857,035	860,319	240.6	857,035	857,035.0	852.54	857,035	862,795.0	10.80
cond-mat	1,633,299	1,637,445	496.13	1,633,299	1,635,776.2	1,488.18	1,642,091	1,647,391.9	10.25

Our algorithm can work on both small budgeted bigger graphs and bigger budgetted small graphs and produced a comparable results to MA and much better results to GR3.

CA-greedy is even faster without the calculation of an objective function. Quadratic nature of the objective function makes it an expensive operation. CA-greedy doesn't need an objective function to produce a solution. Therefore,

Table 10. Comparison with greedy 3. ($b = 10$, $k = 3$)

Networks	opt	$b = 10$							
		GR3		CA-evo			CA-greedy		
		OFV	time	OFV	avg	time	OFV	avg	time
karate	6	8	0.00	6	6.4	0.08	12	12.0	0.06
dolphins	335	340	0.02	335	341.0	0.39	381	408.1	0.28
lesmis	160	160	0.01	160	162.4	0.41	201	212.4	0.19
LindenStrasse	1,151	1,151	0.14	1,151	1,151.7	1.14	1,195	1,247.6	0.29
polbooks	1,715	1,867	0.11	1,715	1,801.6	1.88	1,993	1,995.1	0.59
adjnoun	2,501	2,501	0.15	2,501	2,508.0	2.26	2,607	2,613.4	0.84
football	4,523	4,590	0.15	4,540	4,556.6	4.34	4,682	4,693.6	0.95
netscience	6,785	7,026	0.84	6,785	6,592.3	13.23	6,995	7,054.3	0.60
jazz	14,216	14,306	0.75	14,216	14,329.3	22.63	14,768	14,828.4	4.25
SmallWorld	4,967	5,011	0.33	4,968	4,972.3	5.88	4,968	5,009.3	1.03
Erdos971	20,240	20,442	1.57	20,240	20,278.2	17.85	21,023	21,066.1	3.18
S.Cerevisae	19,861	19,861	8.57	19,861	19,870.6	16.46	20,005	20,083.5	1.30
USAir97	19,157	19,628	0.95	19,157	19,499.9	26.38	20,482	20,550.6	3.45
power	48,602	51,782	136.21	48,602	48,618.5	111.53	49,753	50,055.4	2.28
H.Pylori	27,807	28,204	3.06	27,807	27,813.6	26.98	28,237	28,349.4	2.64
Harvard500	8,581	9,951	1.12	8,581	8,701.6	14.98	8,581	9,011.1	1.51
homer	24,882	24,882	1.39	24,892	24,948.1	24.29	25,146	25,159.8	2.87
celegans_metabolic	25,556	26,830	2.10	25,556	25,938.5	27.38	25,881	25,881.0	3.41
email	241,128	241,144	11.74	241,128	241,143.6	257.88	242,512	243,104.5	9.40
hep-th	321,486	323,268	268.53	321,486	321,653.1	416.21	326,868	327,632.1	5.14
PGPgiantcompo	744,908	769,350	795.27	744,908	745,393.0	998.58	744,908	751,046.5	9.35
cond-mat	1,541,815	1,561,855	1,709.02	1,541,815	1,541,959.0	1,543.94	1,550,187	1,557,413.0	10.18

it can be used effectively to warm start an optimal solution calculations where as GR3 take too long to produce a solution since it repeatedly calls the objective function.

4. CONCLUSIONS AND FUTURE WORK

By exploiting the community structure of complex networks, we have proposed a new destruction strategy, called the collective attacker. Then, we have proposed a scalable greedt algorithm using the collective attacker. In addition, a simple evolutionary algorithm with a novel represention has been investigated and compared with recently proposed two algorithms. The proposed evolutionary algorithm dominates others on the most instances used in the experiments. We calculate betweenness centrality repeatedly and use only top $\sqrt{|V|}$-nodes, and therefore, a specific estimation method for this type of application of betweenness centrality is needed to investigate.

Acknowledgements

The research has received funding from the National University of Mongolia under grant agreement P2020-3978.

Data and code availability statement

The source code and data are available at `https://github.com/zzz/yyy`.

References

[1] A. Arulselvan, C. W. Commander, P. M. Pardalos, O. Shylo, Managing network risk via critical node identification, Risk management in telecommunication networks, Springer (2007) 79–92.
[2] D. Santos, A. de Sousa, P. Monteiro, Compact models for critical node detection in telecommunication networks, Electronic Notes in Discrete Mathematics 64 (2018) 325–334.
[3] V. Cantillo, L. F. Macea, M. Jaller, Assessing vulnerability of transportation networks for disaster response operations, Networks and Spatial Economics 19 (1) (2019) 243–273.
[4] S. Sarker, A. Veremyev, V. Boginski, A. Singh, Critical nodes in river networks, Scientific reports 9 (1) (2019) 1–11.
[5] A. S. Alayande, N. Nwulu, A novel approach for the identification of critical nodes and transmission lines for mitigating voltage instability in power networks, African Journal of Science, Technology, Innovation and Development 11 (3) (2019) 383–390.
[6] L. Tian, A. Bashan, D.-N. Shi, Y.-Y. Liu, Articulation points in complex networks, Nature communications 8 (1) (2017) 1–9.
[7] M. Kammer-Kerwick, N. Busch-Armendariz, M. Talley, Disrupting illicit supply networks: New applications of operations research and data analytics to end modern slavery, Tech. rep., Bureau of Business Research (2018).

[8] U. R. Sumaila, D. Zeller, L. Hood, M. Palomares, Y. Li, D. Pauly, Illicit trade in marine fish catch and its effects on ecosystems and people worldwide, Science advances 6 (9) (2020) eaaz3801.
[9] V. Boginski, C. W. Commander, Identifying critical nodes in protein-protein interaction networks, in: Clustering challenges in biological networks, World Scientific, 2009, pp. 153–167.
[10] A. Veremyev, O. A. Prokopyev, E. L. Pasiliao, Critical nodes for distance-based connectivity and related problems in graphs, Networks 66 (3) (2015) 170–195.
[11] G. U. Alozie, A. Arulselvan, K. Akartunalı, E. L. Pasiliao Jr, Efficient methods for the distance-based critical node detection problem in complex networks, Computers & Operations Research 131 (2021) 105254.
[12] G. U. Alozie, A. Arulselvan, K. Akartunalı, E. L. Pasiliao Jr, A heuristic approach for the distance-based critical node detection problem in complex networks, Journal of the Operational Research Society 73 (6) (2022) 1347–1361.
[13] H. Salemi, A. Buchanan, Solving the distance-based critical node problem, INFORMS Journal on Computing.
[14] A. Arulselvan, C. W. Commander, L. Elefteriadou, P. M. Pardalos, Detecting critical nodes in sparse graphs, Computers & Operations Research 36 (7) (2009) 2193–2200.
[15] R. Aringhieri, A. Grosso, P. Hosteins, R. Scatamacchia, Local search metaheuristics for the critical node problem, Networks 67 (3) (2016) 209–221.
[16] B. Addis, R. Aringhieri, A. Grosso, P. Hosteins, Hybrid constructive heuristics for the critical node problem, Annals of Operations Research 238 (1) (2016) 637–649.
[17] M. Lalou, M. A. Tahraoui, H. Kheddouci, Component-cardinality-constrained critical node problem in graphs, Discrete Applied Mathematics 210 (2016) 150–163.
[18] R. Aringhieri, A. Grosso, P. Hosteins, R. Scatamacchia, A general evolutionary framework for different classes of critical node problems, Engineering Applications of Artificial Intelligence 55 (2016) 128–145.
[19] Y. Zhou, J.-K. Hao, F. Glover, Memetic search for identifying critical nodes in sparse graphs, IEEE transactions on cybernetics 49 (10) (2018) 3699–3712.
[20] M. Garey, D. Johnson, L. Stockmeyer, Some simplified np-complete graph problems, Theoretical Computer Science 1 (3) (1976) 237–267. doi:https://doi.org/10.1016/0304-3975(76)90059-1.
URL https://www.sciencedirect.com/science/article/pii/0304397576900591
[21] B. Addis, M. Di Summa, A. Grosso, Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth, Discrete Applied Mathematics 161 (16-17) (2013) 2349–2360.
[22] M. Lalou, M. A. Tahraoui, H. Kheddouci, The critical node detection problem in networks: A survey, Computer Science Review 28 (2018) 92–117.
[23] L. C. Freeman, A set of measures of centrality based on betweenness, Sociometry (1977) 35–41.
[24] M. Girvan, M. E. Newman, Community structure in social and biological networks, Proceedings of the national academy of sciences 99 (12) (2002) 7821–7826.
[25] M. E. Newman, Communities, modules and large-scale structure in networks, Nature physics 8 (1) (2012) 25–31.
[26] J. Pfeffer, K. M. Carley, k-centralities: Local approximations of global measures based on shortest paths, in: Proceedings of the 21st International Conference on World Wide Web, 2012, pp. 1043–1050.
[27] S. Fortunato, M. E. Newman, 20 years of network community detection, Nature Physics 18 (8) (2022) 848–850.
[28] N. Fan, P. M. Pardalos, Robust optimization of graph partitioning and critical node detection in analyzing networks, in: International Conference on Combinatorial Optimization and Applications, Springer, 2010, pp. 170–183.
[29] D. Purevsuren, G. Cui, M. Qu, N. H. Win, Hybridization of grasp with exterior path relinking for identifying critical nodes in graphs, IAENG International Journal of Computer Science 44 (2) (2017) 157–165.
[30] M. Newman, Networks, Oxford university press, 2018.
[31] D. Sudholt, Crossover is provably essential for the ising model on trees, in: Proceedings of the 7th annual conference on Genetic and evolutionary computation, 2005, pp. 1161–1167.
[32] C. L. Staudt, A. Sazonovs, H. Meyerhenke, Networkit: A tool suite for large-scale complex network analysis, Network Science 4 (4) (2016) 508–530.
[33] V. Batagelj, A. Mrvar, Pajek datasets (2006).
[34] P. Sanders, C. Schulz, D. Wagner, Benchmarking for graph clustering and partitioning, Encyclopedia of social network analysis and mining. Springer.
[35] S. P. Borgatti, M. G. Everett, L. C. Freeman, Ucinet for windows: Software for social network analysis, Harvard, MA: analytic technologies 6 (2002) 12–15.