Scalable heuristic algorithm for identifying critical nodes in networks

Dalaijargal Purevsuren[a,1,*], Gantulga Gombojav[a,1,*]

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

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 complex networks have not yet been extensively addressed in the literature. Specifically, the community structure of networks needs to be considered in more detail. In this study, we propose a critical region (CR) detection method by extracting the community structure, one of the main structural properties of networks. The proposed CR detection method enables us to develop a scalable greedy heuristic for the DCNP. In addition, a simple evolutionary algorithm with a novel representation obtained by the proposed method is investigated. We tested our algorithms on well-known networks and compared them with two recently proposed other algorithms. The proposed greedy heuristic works orders of magnitude faster than the other methods and provides comparable solution quality. Our evolutionary algorithm finds the optimal values on most tests.

Keywords:
scalable algorithms, combinatorial optimization, evolutionary algorithm, critical node problem, community structure, critical region, betweenness centrality

[*]Corresponding author at: Department of Information and Computer Sciences, School of Engineering and Applied Sciences, National University of Mongolia, Ulaanbaatar 14201, Mongolia. E-mail addresses: dalaijargal@gmail.com (D. Purevsuren), gantulgag@seas.num.edu.mn (G. Gombojav)

[1]These authors contributed equally.

1. INTRODUCTION

Nodes in networks have different roles. Identifying critical (key) nodes (players) has practical applications in many areas, including managing telecommunication networks [1, 2], planning effective vaccination strategies [1], assessing the vulnerability of transportation networks to disasters [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 networks [7, 8], and 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 b on the number of nodes to remove, the distance-based critical node problem (DCNP) [10, 11, 12, 13] identifies a set of b vertices whose removal minimizes the pairwise connectivity within D hops in the remaining network. The mathematical formulation can be given as an optimization problem with Equations (1) and (2), where $f(S)$ is the objective function, S is the set of nodes to be deleted from G.

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

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

The critical node problem (CNP), which was first introduced in [1, 14], has several variants depending on their connectivity measures, 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 CNPs [15, 16, 12, 13]. In [15, 16, 12], the algorithms use betweenness centrality (BC) to rank nodes to find which ones to remove. The authors of [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 a centrality measurement as in [23] but can also be used in community detection [24, 25].

Networks are organized with communities [24, 26], in which subnetworks 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 intercommunity edges with exact edge betweenness centrality and removes the most central edge. The GN algorithm repeats the removal of the most central edge. As a result, the communities remain in the end. Note that betweenness centrality can be computed on nodes or edges, and betweenness centrality on nodes is just called betweenness centrality (BC), whereas betweenness centrality on edges is called edge betweenness centrality (edge BC).

The betweenness centrality [23] computes a node's centrality based on how many times the node lies on the paths between all other pairs of nodes. The exact computation of BC

needs to solve the single source shortest paths (SSSP) problem from each node [27], which has $O(n \cdot (n + m)) = O(n \cdot m)$ time complexity, solving n bread-first-search (BFS). The exact computation of BC is not practical on large networks, particularly an application of repeated computation of BC. The estimated value of BC may be an acceptable solution for some problems, including CNPs. An estimation of BC proposed in [28], popular in network analysis, needs to solve the SSSP problem only from some random nodes (called pivots) [29, 30, 28], which has $O(r \cdot (n + m))$ time complexity, solving r bread-first-search, where r is the number of pivots. BC has a version that considers the distance between nodes in BC computation [31], and we call it d-BC. The d-BC computes BC using only paths with at most a given length. Thus, the time complexity of d-BC computation is at most that of BC computation. We combine the d-BC algorithm [31] with the BC estimation technique defined in [30, 28] and call it the estimated d-BC.

Although many approaches study network centralities for CNPs, the community structure has not yet been investigated in the literature on CNPs, except for addressing the application of CNPs for community detection [32, 33, 17]. To the best of our knowledge, this work is the first attempt to integrate community structure with an approach to solving the DCNP. Also, in our work, the d-BC is used to capture distance information instead of ordinary BC.

We propose a critical region (CR) detection method based on the community structure of networks. To do this end, we extend the idea of the GN algorithm by removing multiple (e.g., $O(\sqrt{n})$ nodes simultaneously (instead of single edges in each iteration). In addition, the estimated d-BC is used to find intercommunity regions rather than the exact edge BC. In other words, the extension of the GN algorithm is twofold: first, to remove multiple

nodes in each iteration, and second, to apply the estimated d-BC to find intercommunity nodes. The proposed CR detection method enables us to develop a scalable greedy heuristic for the DCNP. The proposed algorithm was 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 representation obtained by the proposed CR detection method 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 a new critical region detection method, the greedy heuristic, and the 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). First, a critical region (CR) detection method is proposed. Then a scalable greedy heuristic is designed for the DCNP. After that, the greedy heuristic is extended with a simple evolutionary algorithm, similar to $ES(\mu + \lambda)$ [34].

Definition 1. Given the current state of a network, the set of k nodes with the highest d-BC is called the critical region (CR).

As mentioned in Section one, we extend the idea of the Girvan-Newman (GN) algorithm [24] by removing multiple nodes based on a single computation of d-BC. We apply this extension of the GN algorithm to identify the nodes in a critical region. The edges and nodes in the intercommunity have high betweenness centralities [24, 35]. The neighbors of the node with the highest d-BC have a high chance of having a high d-BC when they are also located in the same intercommunity region. Therefore, the top k nodes with the highest d-BC may be located in the same region. An illustration of d-BC ($D = 3$) and a critical region on a sample network are given in Figure 1. After the removal of the CR with $k = 2$ will divide the network into two components (see Figure 1 (B)).

Although removing all the nodes in the critical region may destruct the network well, this removal may involve nodes that are not vital in the destruction, especially if a larger value of k is used in the critical region detection. If the structural difference in the network after restoring a node in a CR is not influential, the node may be reinserted into the network. Reinserting some nodes into the network is called a reinsertion algorithm. A

reinsertion algorithm can be used in two ways: re-evaluating the nodes in a critical region and repairing a constructed solution of the DCNP. These procedures are described in Algorithm 1. Note that we denote the set of nodes in a critical region by "CR".

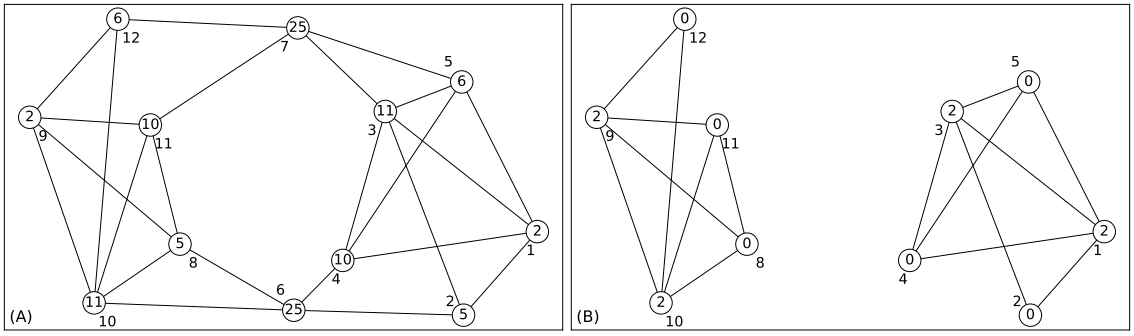

Figure 1. d-BC on a sample network, where the number inside the node is d-BC, and the number beside the node is the node's label. (A) A network has two communities; the nodes with the highest d-BC are nodes 6 and 7, which locate in an intercommunity. (B) the d-BC on the remaining network after removing the CR with $k = 2$.

Algorithm 1 CR-extraction

CR-EXTRACT(G, k)

1: Calculate the d-BC of a given G
2: S = {getting k nodes with the highest d-BC in V}
3: $G = G([V \backslash S])$
4: Call a reinsertion (optional)
5: **return** G, S

2.1. Scalable greedy heuristic for the DCNP

This subsection presents a scalable greedy heuristic using the critical region (CR) extraction algorithm (Algorithm 1). The iterative call of the CR-extraction with a smaller $k = \sqrt{b}$ is investigated here. Removing a CR makes the network vulnerable to destruct or dissect it. If previous CR-extraction has not yet distracted enough, the nodes in the vulnerable region obtain high d-BC scores in the next CR-extraction call. This process

continues until the total number of nodes extracted by the CR-extraction exceeds budget b. We do not use reinsertion in the CR-extraction (Line 4 in Algorithm 1) because we choose a smaller k ($\sqrt{b}$) in critical region detection. The set of nodes extracted by sequential CR-extraction calls is finally examined with a reinsertion algorithm (Line 5 in Algorithm 2). These procedures are given in Algorithms 2 and 3 in detail.

After constructing a solution with the CR-extraction, we apply a reinsertion algorithm to repair the solution. The reinsertion algorithm ranks nodes from low to high impact in destruction according to the following H (Equation 3) function.

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

It measures the destruction quantity of a node u to be restored from the set S, where C is the set of connected components in the $G[V \setminus (S \setminus u)]$ graph, and h is the size of the components in C. A lower value of H means that the set of nodes S cuts graph G into smaller sizes of connected components since it is computed by squaring the size of connected components in the remaining graph. After restoring node u, a higher value of H implies that node u was an essential member of S.

Algorithm 2 CR-greedy algorithm

CR-GREEDY(G, b)
1: $S = \emptyset$
2: **while** $|S| < b$
3: $\quad [G, S'] = CR - extract(G, \sqrt{b})$
4: $\quad S = S \cup S'$
5: reinsert(G, S, b)
6: **return** S

Now let us examine the time complexity of the CR-greedy algorithm. The CR-greedy

Algorithm 3 Reinsertion algorithm

REINSERT(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 d-BC on G
5: Delete top L nodes with the highest d-BC from G and insert them into S
6: **return** S

constructs a solution by calling the CR-extraction algorithm sequentially, not computing the objective function value (OFV) during construction. The cost of the CR-extraction only depends on the computation of the d-BC (Line 1 in Algorithm 1). We use the estimated d-BC with $25 \cdot log^2(n)$ pivots. The time complexity of the estimated d-BC becomes $O((n + m) \cdot log^2(n))$ [29],[28] which is the product of $O(n+m)$ (time complexity of BFS algorithm for computing shortest paths) and $O(log^2(n))$ pivots. Let us consider the upper bound on the number of calls of the CR-extract in Algorithm 2.

Lemma 1. The total number of iterations in Algorithm 2 has an upper bound of $O(\sqrt{b})$.

Proof. Let $m_1, m_2, m_3, \ldots, m_t$ denote the size of CRs in iterations, where t is the total number of the CR-extract calls in Algorithm 2. Therefore, we will obtain Equation 4, which comes from the fact that the total number of removed nodes is less than $b + m_t$.

$$\sum_{i=1}^{t} m_i \leq b + m_t \tag{4}$$

The number of nodes removed in a single iteration equals $\sqrt{b}$, reminding no reinsertion in the CR-extraction. Thus, Equation 4 can be rewritten in Equation 5 by replacing m_i with $\sqrt{b}$. We obtain Equation 6 and Equation 7 with simple calculations. This completes the

proof □.

$$\sum_{i=1}^{t} \sqrt{b} \leq b + \sqrt{b} \tag{5}$$

$$t \cdot \sqrt{b} \leq b + \sqrt{b} \tag{6}$$

$$t \leq \sqrt{b} + 1 \tag{7}$$

The time complexity of the reinsertion algorithm is defined by the estimated d-BC, the most expensive part. Hence, the time complexity of Algorithm 2 is $O(\sqrt{b} \cdot (n+m) \cdot log^2(n))$. When one needs a good solution in a short time and does not require the exact value of the objective function, the CR-greedy returns a solution in $O(\sqrt{b} \cdot (n+m) \cdot log^2(n))$ time. When the budget b is considered as a constant, similar to some papers [13] in the literature, the time complexity of Algorithm 2 is $O((n+m) \cdot log^2(n))$. The DCNP's objective function, which has $O(n \cdot m)$ complexity, is only computed to know how good the solution constructed by the CR-greedy is. If the objective function is computed, its time complexity dominates. Thus, the time complexity becomes $O(n \cdot m)$.

2.2. Evolutionary algorithm for the DCNP

This subsection presents a simple evolutionary algorithm for the DCNP (called the CR-evo). As mentioned before, we denote the set of nodes in a critical region by "CR". In the CR-greedy, the solution is constructed with the first $t = O(\sqrt{b})$ CRs with sequential CR-extraction calls ($S = CR_1 \cup CR_2 \cup \cdots \cup CR_t$, called the elite-t). Let us consider a case in which one CR ($CR_x, 1 <= x <= t$) is missing from the elite-t. In this case, the number of nodes in the $S - CR_x$ becomes less than b, and therefore, we can add more nodes with a

CR-extraction call. The new CR added is different from the missing CR. The elite-t with the new CR can be an interesting solution for further investigation. In general, we can consider many other cases in which many CRs are missing from the elite-t, and new CRs are added to compensate for them. To investigate the CR-based DCNP solutions in detail, we propose an elitist $(\mu + \lambda)$-EA with a CR-based representation. The EA would help to investigate more combinations of CRs rather than consider only CRs in the elite-t. The CR-evo has no mutation operator. It has several building blocks: representation, initialization, recombination, selection, repair operator, and termination condition. We set the parental (μ) and offspring (λ) population size by 3. Algorithms 4, 5, and 6 describe the CR-evo in detail. Note that the CR-evo uses evolutionary algorithm terminology as follows: a population has a number of individuals; an individual chromosome (or individual) has a number of genes; a gene consists of a number of nodes.

The set of nodes obtained by the CR-extraction is used as the representation of the chromosome. An individual (chromosome) has a predefined number of genes (chromosome length). We denote the chromosome length by CL, which is computed by the formula, $CL = \max(c_1 \cdot \sqrt{b} \cdot \log_{10}(n), 5)$, where c_1 is a constant. The size of a gene (GS) is $c_2 \cdot \sqrt{b}$, where c_2 is a constant.

Initialization is an important part of the good performance of the CR-evo, according to the preliminary experiments. At first, this step collects CRs by calling the CR-extraction until the network becomes small enough. We call these collected CRs the elite genes. Note that the elite-m collects CRs until the total number of nodes in the elite-m exceeds the budget b. Individuals are generated by selecting genes from the elite genes at random. Note that the number of elite genes may be less than the length of chromosomes. In other

words, many replications of a gene may occur in an individual.

Recombination consists of two steps: crossover and repair operator. Recombination generates an offspring population with λ offspring. Crossover creates a new offspring from two randomly selected individuals. Specifically, the crossover selects CL genes at random from the union of genes of its parents. If the number of nodes in the new offspring is not equal to budget b, a repair operator is applied to the offspring. Note that each individual has the same chance to be selected for mating. The repair operator (RO) also uses the CR-extraction algorithm if the number of nodes in the offspring is less than b. In the repair step, the CR-extraction is called on the remaining network after removing all nodes in the offspring. This extraction process continues until the number of nodes in the offspring exceeds the budget. When the nodes in the offspring exceed the budget, the reinsertion algorithm (Algorithm 3) is applied to repair it. The CR-evo checks two termination conditions, total running time (T_{max}) and the maximum number of generations with no improvement on the best individual ($idle_{max}$). The latter condition helps the algorithm avoid unnecessarily running.

We use a deterministic (μ+λ) selection method [36]. The best μ individuals (ordered by the objective function value in Equation 1) among the union pool of parental and offspring populations survive for the next generation.

Algorithm 4 CR-evo algorithm

CR-EVO($G, b, T_{max}, idle_{max}$)

$\mu = \lambda = 3$
$P = \text{Init}(G, \mu)$
$idle = 0$
while ($t < T_{max}$ and $idle < idle_{max}$):
    $P' = \text{recombination}(P, b, \lambda)$
    $P = \text{selection}(P, P', \mu)$
    update $idle$
    $t = time_{CPU}()$
return the best individual in P

Algorithm 5 Recombination

RECOMBINATION(P, b, λ)

$M = \emptyset$
$i = 0$
while ($i < \lambda$):
    $s' = \text{crossover}(P)$
    $s = \text{repair-operator}(s', b)$
    $M = M \cup s$
    $i = i + 1$
return M

Algorithm 6 Selection

SELECTION(P, P', μ)

$P'' = P \cup P'$
assign the best μ individuals in P'' to P
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 [12] and [13]. The algorithm presented in [12] is based on a memetic search, and we call this algorithm a "memetic algorithm" (MA). The algorithm presented in [13] is based on greedy 3[16], and we call this algorithm "greedy 3" (GR3). We obtained the MA's source code from the authors, and the link to download the source code of the GR3 is available online.

3.1. Experimental setup

We used the Networkit graph library configured to use a single thread and developed our code in Python 3 [37]. We changed the source code of Networkit to integrate the distance D in BC for computing d-BC. Our experiments were run on a normal desktop computer with Intel(R) Core(TM) i7-7700 CPU @ 3.6Ghz. We used a time limit of $T_{max} = 3,600$ seconds for all executions.

We call the datasets used in [12] and [13] "Benchmark 1" and "Benchmark 2", respectively. We used the same real-world datasets used in these works. These datasets were downloaded from the Pajek dataset [38], 10^{th} DICMACS Implementation Challenge [39] and the UCINET software datasets [40]. We remapped the node indices and formatted these datasets, making them 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 the Github repository. Some characteristics of the networks in Benchmark 1 and Benchmark 2 are shown in Table 1 and Table 2. Benchmark 1 has 14 small-sized networks: the smallest network is *Hi-tech* (33 nodes and 91

edges), and the largest network is *netscience* (379 nodes and 2,216 edges). The networks in Benchmark 2 are 22 small and medium-sized networks: the smallest among them is the *karate* social network with 34 nodes and 78 edges, and the largest among them is the *cond-mat* collaboration network with 16, 726 nodes and 47,597 edges. Twelve networks in Benchmark 2 are larger than those in Benchmark 1, starting from *Erdos971*. The last four networks are significantly larger than the rest.

To evaluate the scalability of methods, we picked eight larger networks having several hundreds of thousands of nodes and millions of edges. They are downloaded from Stanford Network Analysis Platform [41]. We converted directed networks to be undirected and formatted the networks. We call these networks Benchmark 3. The largest network in Benchmark 3 is *web-Stanford* with 281,903 nodes and 2,312,497 edges. Some characteristics of the networks are shown in Table 3.

Table 1. Some characteristics of networks in Benchmark 1

Network	n	m	average degree	Diameter
Hi-tech	33	91	2.76	5
karate	34	78	2.29	5
mexican	35	117	3.34	4
Sawmill	36	62	1.72	8
chesapeake	39	170	4.36	3
dolphins	62	159	2.56	8
Lesmiserable	77	254	3.30	5
Santafe	118	200	1.69	12
Sanjuansur	75	155	2.07	7
Attiro	59	128	2.17	8
LindenStrasse	232	303	1.31	13
SmallWorld	233	994	4.27	4
netscience	379	914	2.41	17
USAir97	332	2,126	6.40	6

3.2. Algorithm configuration

The CR-greedy does not require any algorithmic parameter. It only takes budget b and the distance limit D. We tried to minimize the parameters of the CR-evo. Our evolutionary

Table 2. Some characteristics of networks in Benchmark 2

Network	n	m	average degree	diameter
karate	34	78	2.29	5
dolphins	62	159	2.56	8
lesmis	77	254	3.30	5
polbooks	105	441	4.20	7
adjnoun	112	425	3.79	5
football	115	613	5.33	4
jazz	198	2,742	13.85	6
LindenStrasse	232	303	1.31	13
SmallWorld	233	994	4.27	4
USAir97	332	2,126	6.40	6
Erdos971	429	1,312	3.06	11
celegans_metabolic	453	2,025	4.47	7
Harvard500	500	2,043	4.09	6
homer	561	1,628	2.90	9
H.Pylori	706	1,392	1.97	9
email	1,133	5,451	4.81	8
S.Cerevisae	1,458	1,948	1.34	19
netscience	1,589	2,742	1.73	17
power	4,941	6,594	1.33	46
hep-th	8,361	15,751	1.88	19
PGPgiantcompo	10,680	24,316	2.28	24
cond-mat	16,726	47,594	2.85	18

Table 3. Some characteristics of networks in Benchmark 3

Network	n	m	n+m	average degree	diameter *
p2p-Gnutella31	62,586	147,892	210,478	2.4	6.7
loc-brightkite_edges	58,228	214,078	272,306	3.7	6
soc-Slashdot0902	82,168	948,464	1,030,632	11.5	4.7
loc-gowalla_edges	196,591	950,327	1,146,918	4.8	5.7
com-amazon.ungraph	334,863	925,872	1,260,735	2.8	15
web-NotreDame	325,729	1,497,134	1,822,863	4.6	9.4
twitter_combined	81,306	1,768,149	1,849,455	21.7	4.5
web-Stanford	281,903	2,312,497	2,594,400	8.2	9.7

* 90-percent effective diameter

algorithm depends on the parameters: chromosome length, gene size, population size, and the maximum number of iterations with no improvement. Chromosome length (CL) defines the number of genes in an individual of the population. All the nodes contained in one's genes are merged and give us a solution for the DCNP. As we mentioned before, a gene size can be changed during evolution with the repair operator. We used a very small population size ($\mu = \lambda = 3$), which makes our algorithm run fast and converge rapidly.

The maximum number of iterations with no improvement ($idle_{max}$) sets the number of iterations 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 requires more time to search when this parameter is set to a larger value. Gene size (GS) defines the number of nodes extracted by the CR-extraction (Algorithm 1). We use the output of the CR-extraction as genes in the evolutionary algorithm. 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 according to the preliminary experiments. The five largest networks in Benchmark 1 with b=0.1n were used in the experiments to tune the CR-evo parameters. Additionally, only the averages are considered to obtain stabilized solution quality and time.

First, we studied the suitable values for CL, as shown in Table 4. This parameter has a direct correlation with the gene size, which in turn depends on the budget b. Therefore, we tried different values for the coefficient c_1 in Equation 8 for the CL value.

$$CL = \max(c_1 \cdot \sqrt{b} \cdot \log_{10} n, 5) \tag{8}$$

In Table 4, c_1 is the coefficient of CL in the equation, column CL is the actual length of the chromosome, and columns avg and time are the averages of the objective function value (OFV) and running time over 30 runs, respectively. Other parameters are set as $idle_{max}$=2000 and $GS = \sqrt{b}$ while we change the c_1 parameter. The running time decreases for some time when c_1 increases, and it increases again at a certain point. The same behavior is observed for the objective function value. When an individual has more genes, the quality of the solution increases. However, the speed of the convergence de-

clines when the length becomes too large. The suitable values for c_1 are 2 and 3, as shown in Table 4. We set $c_1 = 2$ for further comparisons as it shows a slightly better solution quality.

Table 4. The performance when the CL parameter changes according to Equation 8. Columns avg and time are the averages of the OFVs and running time over 30 runs, respectively. c_1 is the coefficient in Equation 8. The other parameters are $GS = \sqrt{b}$ and $idle_{max} = 2000$ in this experiment

Networks	b	$c_1 = 0.5$			$c_1 = 1$			$c_1 = 2$		
		CL	avg	time	CL	avg	time	CL	avg	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

Second, we studied the effect of gene size (GS) on the solution quality and running time. GS is the number of nodes in a single gene that is obtained by the CR-extraction. We used Equation 9 for GS size and iterated over several values for the c_2 coefficient to analyze this parameter.

$$GS = c_2 \cdot \sqrt{b} \tag{9}$$

Table 5 shows that the running time increases when GS is small and decreases as it grows. It requires more iterations in the initialization step of the algorithm and a longer chromosome when the number of nodes contained in a gene is small. Thus, it slows down the algorithm. Although larger values of GS increase the algorithm speed, the solution quality decreases. The experiment shows that the sweet spot for GS is c_2=1.

Finally, we experimented with the $idle_{max}$ parameter on four different values in Table 6

Table 5. Performance when GS is changed according to Equation 9. avg and time are the averages of the objective function values and running time over 30 runs, respectively. c_2 is the coefficient in Equation 9. The other parameters are $idle_{max} = 2000$ and $c_1 = 2$ in this experiment.

Networks		$c_2 = 0.5$			$c_2 = 1$			$c_2 = 1.5$			$c_2 = 2$		
	b	GS	avg	time	GS	avg	time	GS	avg	time	GS	avg	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	1,706.5	84.86	5	1,700.6	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	3,295.0	202.49	6	3,307	181.27	21	3,329.0	180.1	27	3,338.8	145.28
Total			6,464.6	371.46		6,462.2	322.32		6,486.0	368.11		6,529.3	309.15

and measured the average solution quality and time over 30 runs. The time required to find the current best solution (column btime) increases as $idle_{max}$ increases. This means that $idle_{max}$ has an effect on the current best solution because it takes more time to improve its result. If btime remains small compared to the time column, the algorithm reaches the best solution early and spends useless cycles to further improve. The column ratio shows this effect by taking the ratio of btime and the actual running time (column time). This ratio degrades as $idle_{max}$ increases and is little a bit bigger on *USAir97* and *netscience*, meaning that they still got a chance to improve when $idle_{max}$ is set to 10000. When $idle_{max} = 5000$ and 10000 are compared, the running time almost doubles in 10000, but there is little difference in the solution averages and no change in the minimum value except for *USAir97*. We obtain comparable results and far faster running time when the parameter is set to 100. We choose $idle_{max} = 100$ and $idle_{max} = 5000$ in the comparisons of our method against the state-of-the-art algorithms considering both the solution quality and running time.

We ran our CR-evo algorithm on Benchmark 1, Benchmark 2, and Benchmark 3 with the parameter configurations shown in Table 7, which are proven to perform well both in running time and solution quality by the experiments in this section.

Table 6. $idle_{max}$ parameter experiment. Avg, btime, and time are the averages of the objective function values, the time required to find the current best solution, and the actual running time over 30 runs respectively. The ratio is the ratio of btime to time. The other parameters are $c_1 = 2$ 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

Table 7. Parameter configuration for the CR-evo

Parameter	**Algorithm configuration**
Length of chromosome (CL)	$\max(2 \cdot \sqrt{b} \cdot \log_{10} n, 5)$
Parental and offspring population size	$\mu = \lambda = 3$
Number of iterations with no improvement ($idle_{max}$)	100 or 5000
Gene size (GS)	$\sqrt{b}$

3.3. Comparison with other methods

We compared the CR-evo and the CR-greedy against the MA and the GR3 on Benchmark 1, Benchmark 2, and Benchmark 3 datasets, respectively. The CR-evo was run with two different $idle_{max}$ parameters to see how well it performed compared with other algorithms in a short amount of time. We denote the CR-evo with $idle_{max} = 100$ and $idle_{max} = 5000$ by the CR-evo-i100 and the CR-evo-i5000, respectively. We executed the MA algorithm on our machine and measured the average of objective function values (OFVs), min of OFVs, and running time over 30 runs. We also executed the GR3 algorithm once on our machine and measured the OFV and the running time. Note that the GR3 is a deterministic algorithm. The optimal values for datasets in Benchmark 1 and Benchmark 2 are available in [12] and [13], and we collect the optimal values from the papers.

3.3.1. Results on Benchmark 1

Table 8 and Table 9, with summaries at their bottom, show the comparisons against the MA on Benchmark 1 with the budget sizes $b = 0.05n$ and $b = 0.1n$, respectively.

First, let us consider the solution quality of the CR-evo and the CR-greedy on Benchmark 1 with budget size $b = 0.05n$ in Table 8. The minimum values obtained by the CR-evo-i100 are optimal for 12 out of 14 networks except *Hi-tech* and *dolphins*. The CR-evo-i5000 produced the same results, but its average solution quality was better than the CR-evo-i100's average solution quality, producing more stable results: it produced 10 best averages, out of which 8 are optimal, whereas the CR-evo-i100 produced 6 best averages, out of which 6 are optimal. The CR-greedy produced 7 optimal minimum values, and the averages of the solutions show that its results are quite stable. The MA produced optimal values on all the networks and produced 11 best averages, out of which 9 are optimal. The CR-evo-i5000 produced better averages on 3 networks, namely *USAir97*, *SmallWorld*, and *LindenStrasse*.

When this budget size is considered, Table 8 suggests that the CR-greedy is approximately 100 to 300 times faster than the CR-evo-i5000 and the MA on larger networks such as *SmallWorld*, *netsicence*, and *USAir97*. Although the speed of the CR-greedy and the CR-evo-i100 is similar on smaller networks, the difference in running time of them grows as the network size increases. The MA is faster than the CR-evo-i5000 on small datasets and gets slower on larger datasets such as *netsicence* and *USAir97*. The CR-evo-i100 gives a very reasonable running time and solution quality. The CR-evo-i100 ran approximately 30 to 50 times faster than the CR-evo-i5000 and was able to produce the same minimum values as the CR-evo-i5000 over 30 runs, and the CR-evo-i100 ran 2 to 37 times faster

than the MA. Note that the budget sizes of the *Hi-tech, karate, mexican, Sawmill*, and *chesapeake* networks are only 1 in Table 8 when b is set to $0.05n$.

Table 8. Comparison with the MA on Benchmark 1. opt indicates the optimal objective function value (OFV). min, avg, and time are the optimal OFV, minimum of OFVs, average of OFVs, and average running time over 30 runs, respectively. Optimal values are in italic, and best values are in bold ($b = 0.05n$, $D = 3$).

Networks	opt	$b = 0.05n$											
		MA			CR-evo-i100			CR-evo-i5000			CR-greedy		
		min	avg	time	min	avg	time	min	avg	time	min	avg	time
Hi-tech	397	***397***	***397.0***	0.14	412	412.0	0.05	412	412.0	2.07	427	427.0	0.04
karate	324	***324***	***324.0***	0.19	***324***	***324.0***	0.05	***324***	***324.0***	1.95	***324***	***324.0***	0.03
mexican	527	***527***	***527.0***	0.23	***527***	***527.0***	0.04	***527***	***527.0***	1.82	***527***	***527.0***	0.04
Sawmill	215	***215***	***215.0***	0.11	***215***	***215.0***	0.02	***215***	***215.0***	1.11	***215***	***215.0***	0.03
chesapeake	696	***696***	***696.0***	0.30	***696***	***696.0***	0.06	***696***	***696.0***	2.18	***696***	***696.0***	0.04
dolphins	820	***820***	**822.3**	1.11	830	838.6	0.13	830	835.1	5.22	849	849.0	0.07
Lesmiserable	930	***930***	***930.0***	1.43	***930***	947.8	0.16	***930***	940.3	6.69	952	952.0	0.06
Santafe	305	***305***	***305.0***	1.22	***305***	***305***.0	0.11	***305***	***305.0***	4.57	***305***	***305.0***	0.03
Sanjuansur	803	***803***	***803.0***	0.71	***803***	804.1	0.12	***803***	***803.0***	5.24	***803***	***803.0***	0.06
Attiro	743	***743***	***743.0***	0.40	***743***	***743.0***	0.12	***743***	***743.0***	4.75	***743***	***743.0***	0.06
LindenStrasse	1,054	***1,054***	1,054.8	4.75	***1,054***	1,055.6	0.39	***1,054***	**1,054.2**	13.71	1,178	1,181.0	0.07
SmallWorld	4,629	***4,629***	4,630.4	33.80	***4,629***	4,644.2	1.61	***4,629***	***4,629.0***	65.04	7,575	7,593.9	0.26
netscience	2,102	***2,102***	**2,104.4**	35.94	***2,102***	2,132.3	0.97	***2,102***	2,104.6	30.83	2,379	2,439.4	0.11
USAir97	10,623	***10,623***	10,718.5	188.82	***10,623***	10,889.7	5.01	***10,623***	**10,634.0**	181.15	11,640	11,672.8	0.59
# of best		14	11		12	6		12	10		7	7	
# of opt		14	9		12	6		12	8		7	7	

The performance evaluations with $b = 0.1n$ are shown in Table 9. Both the CR-evo-i100 and the CR-evo-i5000 produced 13 optimal minimum values and produced 6 and 9 best averages, respectively. They produced optimal objective function values on all the networks except *USAir97*. The MA produced 13 optimal values and 6 best averages. The CR-evo-i5000 produced more stable results than the MA, producing 2 more best averages. The CR-evo-i100 produced better averages on *karate*, *chesapeake*, *dolphins*, *Sanjuansur*, and *Attiro* networks compared to the MA. The CR-greedy produced four optimal minimum values and four optimal average values, and its averages stayed close to the minimum values.

The running time of the MA increases noticeably compared to Table 8, whereas the

running time of the CR-evo-i100, the CR-evo-i5000, and the CR-greedy stays stable as the budget increases from $0.05n$ to $0.1n$. We discuss the budget sensitivity of the algorithms in subsection 3.3.4.

Table 9. Comparison with the MA on Benchmark 1. opt indicates the optimal objective function value. min, avg, and time are minimum of OFVs, average of OFVs, and average running time over 30 runs, respectively. Optimal values are in italic, and best values are in bold ($b = 0.1n$, $D = 3$).

Networks	opt	$b = 0.1n$											
		MA			CR-evo-i100			CR-evo-i5000			CR-greedy		
		min	avg	time	min	avg	time	min	avg	time	min	avg	time
Hi-tech	293	***293***	294.2	0.42	***293***	297.8	0.06	***293***	**293.8**	3.43	302	315.6	0.03
karate	147	***147***	162.6	0.29	***147***	***147.0***	0.03	***147***	***147.0***	1.71	***147***	***147.0***	0.02
mexican	358	***358***	***358.0***	0.44	***358***	***358.0***	0.06	***358***	***358.0***	2.67	***358***	***358.0***	0.04
Sawmill	135	***135***	135.3	0.19	***135***	***135.0***	0.03	***135***	***135.0***	2.41	***135***	***135.0***	0.01
chesapeake	512	***512***	514.7	0.79	***512***	***512.0***	0.07	***512***	***512.0***	4.38	***512***	***512.0***	0.05
dolphins	583	***583***	592.4	1.89	***583***	**583.6**	0.14	***583***	583.7	6.59	628	635.2	0.06
Lesmiserable	323	***323***	***323.0***	2.40	***323***	***323.0***	0.11	***323***	***323.0***	6.49	355	355.0	0.04
Santafe	116	***116***	***116.0***	2.42	***116***	116.1	0.11	***116***	116.1	4.14	119	119.0	0.02
Sanjuansur	457	***457***	458.2	1.70	***457***	457.5	0.16	***457***	**457.4**	5.01	550	550.0	0.05
Attiro	444	***444***	446.2	0.71	***444***	444.3	0.11	***444***	***444.0***	4.46	465	465.5	0.04
LindenStrasse	429	***429***	**434.0**	11.01	***429***	439.1	0.35	***429***	435.6	14.16	455	465.3	0.05
SmallWorld	1,694	***1,694***	1,698.5	47.19	***1,694***	1,737.8	1.08	***1,694***	**1,696.8**	43.22	1,868	2,064.1	0.21
netscience	897	***897***	**905.3**	92.29	***897***	913.8	0.83	***897***	904.8	37.71	1,029	1,034.1	0.09
USAir97	3,100	**3,108**	**3,277.6**	272.98	3,201	3,346.6	3.33	3,201	3,309.6	119.76	3,313	3,446.6	0.42
# of best		14	6		13	6		13	9		4	4	
# of opt		13	3		13	5		13	6		4	4	

3.3.2. Results on Benchmark 2

The CR-evo and the CR-greedy are compared against the GR3 on the Benchmark 2 dataset with budget sizes of $b = 5$ and $b = 10$ in Table 10 and Table 11, respectively. These budget sizes were chosen with respect to the performance evaluation of the GR3 published in [13]. We ran the CR-evo with only $idle_{max} = 100$ parameter here since the CR-evo-i100 and the CR-evo-i5000 produced the same minimum values. GR3 has no average values because it's a deterministic algorithm. Therefore, only the OFV and the running time are measured. The performances of the algorithms are summarized at the bottom of the tables.

In Table 10 with $b = 5$, the CR-evo-i100 produced 22 best minimum values, out of which 20 are optimal. The CR-evo-i100 performs well when this budget is considered, producing optimal results on all the networks except *polbooks* and *homer*. It produced 14 best averages, out of which 5 are optimal. The CR-greedy produced 6 minimum values, which are all optimal and produced one optimal average. GR3 produced 12 best objective values, out of which 11 are optimal. Note that the OFV of GR3 is compared with the averages of the CR-evo-i100 and the CR-greedy in terms of determining the best average values.

Table 11 with $b = 10$ shows that the CR-evo-i100's solution quality stayed almost the same, producing 22 best minimum values, out of which 19 are optimal. It could not produce the optimal values on *football*, *SmallWorld*, and *homer*. Both the GR3 and the CR-greedy showed decreased solution quality, producing 5 and 2 best minimum values, respectively, as the budget increased. The CR-evo-i100 produced 14 best averages and 1 optimal average on *lesmis*, whereas the CR-greedy produced 1 best average.

The CR-greedy is the fastest algorithm among them, and the GR3 is the second fastest algorithm according to Table 10 and Table 11. Although the CR-greedy is slower than the GR3 on some small networks, the CR-greedy's running time growth is much slower than the GR3 as the network size increases. The CR-greedy ran approximately 10 to 20 times faster than the GR3 on the larger networks such as *hep-th, PGPgiantcompo*, and *cond-mat* when the budget is 5 (see Table 10). When the budget is 10, the CR-greedy runs approximately 30 to 45 times faster than the GR3 on the larger networks. The reason is that the running time of the GR3 is doubled on most networks as the budget increases from 5 to 10. The CR-evo-i100 is approximately 2 to 15 times slower than the MA when the

budget is 5. The ratio becomes 1 to 6 as the budget increases from 5 to 10.

Table 10. Comparison with the GR3 on Benchmark 2. opt indicates the optimal objective function value (OFV). min, avg, and time are minimum of OFVs, average of OFVs, and average running time over 30 runs, respectively. Optimal values are in italic, and best values are in bold. ($b = 5$, $D = 3$).

Networks	opt	$b = 5$							
		GR3		CR-evo-i100			CR-greedy		
		OFV	time	min	avg	time	min	avg	time
karate	41	***41***	0.00	***41***	***41.0***	0.03	65	65.0	0.01
dolphins	662	678	0.01	***662***	**663.3**	0.13	685	688.7	0.06
lesmis	517	535	0.01	***517***	**519.7**	0.13	***517***	527.2	0.05
polbooks	2,555	2,673	0.05	**2,578**	**2,612.9**	0.48	2,661	2,663.2	0.14
adjnoun	3,719	***3,719***	0.05	***3,719***	3,722.8	0.65	4,059	4,060.2	0.18
football	5,362	***5,362***	0.10	***5,362***	5,384.1	0.98	5,466	5,469.4	0.20
jazz	16,136	16,185	1.16	***16,136***	16,186.1	4.28	16,922	16,939.4	0.94
LindenStrasse	1,810	1,815	0.04	***1,810***	**1,812.3**	0.32	1,837	1,841.5	0.08
SmallWorld	6,964	***6,964***	0.21	***6,964***	***6,964.0***	1.74	***6,964***	7,288.0	0.21
USAir97	29,486	***29,486***	1.16	***29,486***	29,669.4	7.45	30,021	30,021.0	0.80
Erdos971	25,737	***25,737***	0.50	***25,737***	25,902.5	5.61	26,136	26,164.9	0.67
celegans_metabolic	44,967	***44,967***	1.23	***44,967***	***44,967.0***	7.21	46,488	46,488.0	0.89
Harvard500	16,448	19,241	0.77	***16,448***	**16,919.8**	4.46	20,368	20,368.0	0.49
homer	45,828	**45,838**	1.06	**45,838**	45,922.5	7.43	46,137	46,137.0	0.83
H.Pylori	37,626	***37,626***	0.82	***37,626***	***37,626.0***	4.78	***37,626***	***37,626.0***	0.69
email	263,409	***263,409***	4.11	***263,409***	263,855.3	59.97	264,756	265,216.4	2.52
S.Cerevisae	25,190	***25,190***	1.24	***25,190***	***25,190.0***	3.52	***25,190***	25,277.4	0.35
netscience	8,390	8,898	0.86	***8,390***	**8,401.6**	2.56	8,412	8,475.5	0.18
power	50,410	52,456	8.51	***50,410***	**50,423.6**	22.55	50,514	50,607.7	0.6
hep-th	345,320	***345,320***	20.73	***345,320***	345,353.5	97.60	***345,320***	346,740.3	1.78
PGPgiantcompo	857,035	860,319	45.81	***857,035***	**857,144.5**	211.56	***857,035***	861,536.5	3.72
cond-mat	1,633,299	1,637,445	95.73	***1,633,299***	**1,635,322.6**	500.11	1,637,445	1,642,051.7	4.89
# of best		12		22	14		6	1	
# of opt		11		20	5		6	1	

3.3.3. Results on Benchmark 3

We ran all five algorithms on networks in Benchmark 3 with a budget size of 1,000 and a time limit of 3,600 seconds. The experimental results are tabulated in Table 12. The MA and the GR3 exceed the time limit on all networks. The CR-evo returns an answer only on *p2p-Gnutella31* and *loc-brightkite_edges* networks within the time limit. The CR-greedy finds a solution on all networks. More specifically, the CR-greedy needs less than 100 seconds on *p2p-Gnutella31* and *loc-brightkite_edges* networks to finish. The longest

Table 11. Comparison with the GR3 on Benchmark 2. opt indicates the optimal objective function value (OFV). min, avg, and time are minimum of OFVs, average of OFVs, and average running time over 30 runs, respectively. Optimal values are in italic, and best values are in bold. (b = 10, D = 3)

Networks	opt	b = 10							
		GR3		CR-evo-i100			CR-greedy		
		OFV	time	min	avg	time	min	avg	time
karate	6	9	0.02	***6***	**6.2**	0.04	***6***	6.5	0.01
dolphins	335	340	0.02	***335***	341.4	0.13	425	428.7	0.04
lesmis	160	***160***	0.02	***160***	***160.0***	0.12	178	178.0	0.03
polbooks	1,715	1,867	0.11	***1,715***	**1,814.2**	0.51	2,128	2,130.9	0.13
adjnoun	2,501	***2,501***	0.13	***2,501***	2,508.6	0.64	2,673	2,680.8	0.16
football	4,523	4,590	0.32	**4,540**	**4,561.5**	1.19	4,681	4,753.2	0.20
jazz	14,216	14,306	3.28	***14,216***	14,351.4	5.24	15,762	15,762.1	0.93
LindenStrasse	1,151	***1,151***	0.14	***1,151***	1,153.2	0.34	1,232	1,287.8	0.07
SmallWorld	4,967	5,011	0.46	**4,968**	**4,984.0**	1.66	5,225	5,238.4	0.18
USAir97	19,157	19,628	2.14	***19,157***	**19,598.9**	6.84	20,777	20,777.0	0.68
Erdos971	20,240	20,442	1.45	***20,240***	**20,269.0**	4.79	21,239	21,326.3	0.63
celegans_metabolic	25,556	26,830	2.53	***25,556***	26,019.5	6.07	25,881	**25,881.0**	0.79
Harvard500	8,581	9,951	2.26	***8,581***	**8,636.8**	3.51	9,998	11,226.1	0.44
homer	24,882	**24,892**	1.69	**24,892**	24,931.2	6.25	25,146	25,146.0	0.74
H.Pylori	27,807	28,204	2.80	***27,807***	**27,870.2**	6.67	28,237	28,589.1	0.61
email	241,128	241,144	10.59	***241,128***	241,163.5	61.25	245,711	245,712.6	2.66
S.Cerevisae	19,861	***19,861***	3.49	***19,861***	19,865.8	3.45	***19,861***	20,140.3	0.34
netscience	6,785	7,026	3.12	***6,785***	**6,807.1**	3.56	7,023	7,026.7	0.21
power	48,602	51,782	21.91	***48,602***	**48,631.4**	32.95	49,237	49,248.6	0.65
hep-th	321,486	323,268	58.78	***321,486***	**321,661.2**	119.53	324,481	325,743.5	2.01
PGPgiantcompo	744,908	769,350	114.86	***744,908***	**746,183.6**	332.07	750,128	751,064.5	3.88
cond-mat	1,541,815	1,561,855	244.39	***1,541,815***	**1,542,408.7**	624.52	1,554,487	1,555,544.7	5.54
# of best		5		22	14		2	1	
# of opt		4		19	1		2	0	

time that the CR-greedy needs to complete is 1,733 seconds on *twitter_combined* network, which has 81,306 nodes and 1,768,149 edges. *twitter_combined* network is ordered second in terms of the summation of the number of nodes and edges ($n + m$) (see Table 3) and the densest having an average degree of 21.7. The quality of solutions obtained from the five algorithms is harder to compare because other methods exceed the time limit. Instead, the results in Table 12 demonstate the scalability of the CR-greedy on larger networks that have up to two million edges.

3.3.4. Running time against budget

The subsection examines the running time of the algorithms as the budget increases. First, we can see their dependence on the budget size by comparing Table 8 with Table 9 and Table 10 with Table 11.

According to the results in Table 8 and Table 9, the running time of the MA grows by a factor of 1.4 to 2.9 as the budget increases. The CR-evo-i100 and the CR-evo-i5000 run even faster on many networks in Benchmark 1 as the budget increases from $0.05n$ to $0.1n$. The reason is that the CR-evo's genes consist of more nodes as the budget grows, which makes it converge faster. For example, The CR-evo-i5000 runs faster by approximately 1.2 factor on larger networks, i.e., *USAir97* and *netsicence*. The running time of the CR-greedy almost stays stable in the budget change.

According to the results in Table 10 and Table 11, the most budget-sensitive algorithm is the GR3, as we can see that its running time of the execution with $b = 10$ (see Table 11) is 2 to 3 times higher on most networks compared to that of the execution with $b = 5$ (see Table 10). On the other hand, the running time of the CR-evo-i100 increased slightly on some networks and decreased on some. The running time of the CR-greedy stays stable as the budget increases from 5 to 10.

Now, let us examine the budget sensitivity of the algorithms in a more detailed view. The running time of the algorithms is measured on *netscience* network (1,589 nodes and 2,742 edges) in Benchmark 2 with five different values ($b = 10$, 45, 80, 115, and 150) of the budget, starting from 10 to 150, which equals roughly $0.1n$. The running time against the abovementioned budget values is plotted in Figure 2. The running time of CR-evo-i100, CR-evo-i5000, and CR-greedy is relatively stable, whereas the running time of the

MA and the GR3 increases sharply as the budget increases (see Figure 2).

Table 12. Experimental results on Benchmark 3. min, avg, and time are the objective function value (OFV), average of OFVs, and average running time over 30 runs, respectively. ($b = 1,000$, $D = 3$).

Networks	$b = 1,000$					
	CR-evo			CR-greedy		
	min	avg	time	min	avg	time
p2p-Gnutella31	7,739,471	7,762,568.8	3,633.77	7,846,028	7,853,516.8	94.41
loc-brightkite_edges	8,529,448	8,823,948.6	3,669.92	11,451,655	11,453,151.8	92.65
soc-Slashdot0902	N/A[1]	N/A	ETL[2]	161,609,452	162,232,401.2	923.47
loc-gowalla_edges	N/A	N/A	ETL	167,981,076	168,878,701.1	664.31
com-amazon.ungraph	N/A	N/A	ETL	14,653,598	14,662,143.9	544.39
web-NotreDame	N/A	N/A	ETL	17,637,705	17,640,149.9	537.8
twitter_combined	N/A	N/A	ETL	140,730,435	140,779,179.8	1,773.09
web-Stanford	N/A	N/A	ETL	26,986,418	26,986,418	733.2

1 N/A stands for not available.
2 ETL stands for exceeding time limit.

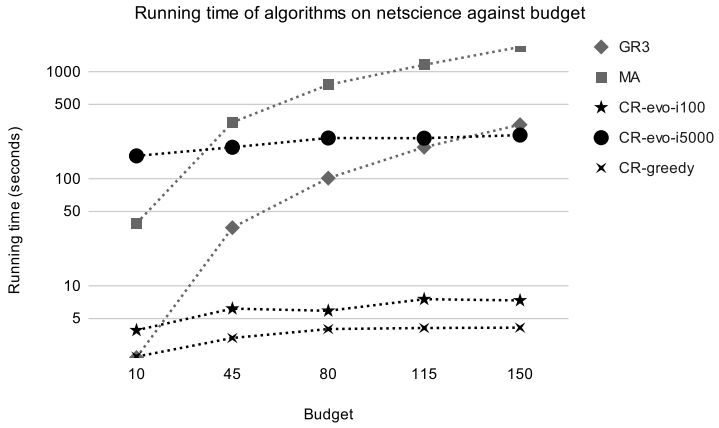

Figure 2. Running time against budget

4. CONCLUSIONS AND FUTURE WORK

By exploiting the community structure of networks, we have proposed a new critical region detection method. Then, we proposed a constructive greedy heuristic for the

distance-based critical node problem. The time complexity of the proposed greedy heuristic is $O(\sqrt{b} \cdot (n+m) \cdot log^2(n))$, where b is the budget. In addition, a simple evolutionary algorithm with a novel representation has been investigated and compared with two recently proposed algorithms. The proposed evolutionary algorithm finds optimal solutions on most datasets used in the experiments in a shorter time. In future work, a careful design of a reinsertion algorithm may improve the performance of the greedy heuristic. We calculate betweenness centrality repeatedly and use only top k-nodes, and therefore, a specific approximation 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. We thank the authors of [12] for sharing their source code with us.

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 8 (4) (2022) 1309–1326.

[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] S. Fortunato, M. E. Newman, 20 years of network community detection, Nature Physics 18 (8) (2022) 848–850.

[27] U. Brandes, A faster algorithm for betweenness centrality, Journal of mathematical sociology 25 (2) (2001) 163–177.

[28] R. Geisberger, P. Sanders, D. Schultes, Better approximation of betweenness centrality, in: 2008 Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX), SIAM, 2008, pp. 90–100.

[29] D. E. J. Wang, Fast approximation of centrality, Graph algorithms and applications 5 (5) (2006) 39.

[30] U. Brandes, C. Pich, Centrality estimation in large networks, International Journal of Bifurcation and Chaos 17 (07) (2007) 2303–2318.

[31] 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.

[32] 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.

[33] 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.

[34] A. Slowik, H. Kwasnicka, Evolutionary algorithms and their applications to engineering problems, Neural Computing and Applications 32 (16) (2020) 12363–12379.

[35] M. Newman, Networks, Oxford university press, 2018.

[36] T. Back, Evolution strategies 1: Variants and their computational implementation, Genetic Algorithms in Engineering and Computer, Science (1995) 127–140.

[37] C. L. Staudt, A. Sazonovs, H. Meyerhenke, Networkit: A tool suite for large-scale complex network analysis, Network Science 4 (4) (2016) 508–530.

[38] V. Batagelj, A. Mrvar, Pajek datasets (2006).

[39] P. Sanders, C. Schulz, D. Wagner, Benchmarking for graph clustering and partitioning, Encyclopedia of social network analysis and mining. Springer.

[40] 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.

[41] J. Leskovec, R. Sosič, Snap: A general-purpose network analysis and graph-mining library, ACM Transactions on Intelligent Systems and Technology (TIST) 8 (1) (2016) 1–20.