Soft Computing
https://doi.org/10.1007/s00500-021-06475-w

APPLICATION OF SOFT COMPUTING

Efficient network dismantling through genetic algorithms

Wei Lin[1] · **Sebastian Wandelt**[2,3] · **Xiaoqian Sun**[1,2,3]

Accepted: 16 October 2021
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2021

Abstract
Throughout the past decades, network dismantling gained an increasing interest by the research community, given tremendous importance of robust socio-technical systems. The problem of optimally dismantling a given network is provably NP-hard. Accordingly, existing studies on network dismantling resort to various heuristics, including the use of node centralities and finely tuned decycling and cutting techniques. However, it is known that these existing techniques largely lack the optimal baseline. Particularly, these techniques perform bad when compared to (expensive-to-compute) betweenness-based attacks. Therefore, there is a strong need for developing scalable, yet accurate attacking methods for being able to understand the robustness of large, real-world complex systems. In this study, we propose a novel network attack technique based on genetic algorithms. In order to develop a scalable framework, we first design an exact method for measuring the effectiveness of an attack, requiring $O(|E|)$ time, where $|E|$ is the number of edges in the network. Since this algorithm runs in linear time of the network size, it can scale up well for very large networks. Second, we develop and analyze a collection of genetic population constructors, which aim at providing a rich set of initial genetic material to the framework. Several genetic operators are proposed, which preferably select previously critical nodes to be attacked first. Finally, we evaluate our framework on a wide range of real-world networks. Results show that our novel technique significantly outperforms the state-of-the-art methods, providing an interesting sweet spot between attack quality and computational complexity. We believe that our work contributes toward the scalable robustness estimation of complex networks, and that the perspective of using non-deterministic methods will inspire future research in this domain.

Keywords Complex networks · Dismantling · Genetic algorithms

1 Introduction

A plethora of real-world systems can be described as complex networks, with nodes representing atomic elements of the system and edges encoding their interactions. Examples for the successful modeling of systems from the complex network perspective include power grids (Albert et al. 2004; Cuadra et al. 2015), transportation systems (Colizza et al. 2006; Zanin and Lillo 2013; Sun et al. 2015), communication infrastructure (Yook et al. 2002; Albert and Barabási 2002), social networks Duijn et al. (2014), and biological systems Merico et al. (2009). Complex networks facilitate the identification of hidden patterns in these systems. One such phenomenon is the robustness of the network against node or link failures. Most networks carry a rather well-understood resilience against random failures, where individual nodes fail with a given probability. Regarding the resilience against targeted attacks, many real-world systems are susceptible to attacks of so-called critical nodes, where the attack to a few critical nodes often leads to a cascade of the whole network, which reduces the network's function significantly Peters et al. (2008). Examples for such cascading effects during the past few years include large-scale power outages in US and Italy (Ash and Newth 2007; Corsi and Sabelli 2004), virus-spreading on the Internet Strogatz (2001), the crucial

✉ Xiaoqian Sun
sunxq@buaa.edu.cn

Wei Lin
louislin@buaa.edu.cn

Sebastian Wandelt
wandelt@buaa.edu.cn

[1] School of General Engineering, Beihang University, 100191 Beijing, China

[2] National Key Laboratory of CNS/ATM, School of Electronic and Information Engineering, Beihang University, 100191 Beijing, China

[3] Beihang Hangzhou Innovation Institute Yuhang, Xixi Octagon City, Yuhang District 310023, Hangzhou, China

Published online: 19 November 2021

role of particular individuals in social networks Lerman and Ghosh (2010), and large-scale failures in transportation systems Brooker (2010). These disruptions have tremendous socio-economical effects on the society Gao et al. (2015). Accordingly, analyzing and understanding the robustness of networks against targeted attacks is an important, yet challenging research problem.

The identification of an optimal attack to a network is computationally hard Braunstein et al. (2016). It can be shown that the underlying node selection/ordering problem is NP-hard, which means that it is unlikely to develop exact techniques which scale up for networks of size beyond a few dozens of nodes. Accordingly, existing methods rely on the use of heuristics, which can be divided into two types. The first type of heuristics are node centralities. Such centralities measure the importance of individual nodes in the network, usually based on some topological properties. For instance, if a node has a large number of neighbors, it can be considered influential (at a local scale). Other centralities are tailored toward global scales, such as betweenness and closeness. It has been shown that betweenness-based attacks are particularly efficient Wandelt et al. (2018); specifically if the betweenness values are recomputed under node removal (so-called interactive attacks). Unfortunately, the computational complexity of interactive betweenness attack is at least cubic in the number of nodes Brandes (2001); the exact complexity depends on the network density and topology. Therefore, the computation of interactive betweenness-based attack is restricted to networks of a few thousands of nodes, whereas in biological or social systems, the number of nodes can easily reach hundreds of millions. Note that, however, the results of this method are not optimal; it is just empirically the best attacking method conceived so far. The second type of methods in the literature are so-called network dismantling methods. These methods have in common that they were designed specifically for the problem of generating network attacks. The concepts used for dismantling vary, including the computation of collective influence Morone et al. (2016), a combination of decycling and tree breaking Braunstein et al. (2016), or the identification of so-called articulation points Tian et al. (2017), whose removal directly reduces the size of the network's largest component. These methods were developed with the intention to compute attacks efficiently for large-scale networks; at least for specific types, e.g., strictly hierarchical networks. However, these methods do not perform well even against interactive betweenness, let alone yielding close-to-optimal results.

In this study, we take a different perspective on the network dismantling problem. Instead of designing a deterministic dismantling algorithm, we propose to use genetic algorithms (GA) Whitley (1994) for computing attacks to the networks. This idea is quite intuitive, given the fact that dismantling is a hard optimization problem. From the perspective of algorithm design, GA is quite different from those in the literature. Most of the existing methods generate the attack sequence by ranking the importance of nodes. These node-ranking-based methods tend to neglect dependencies among nodes. GA not only considers a single node, but focuses more on the attack sequence composed of multiple nodes. In the course of GA, the target network will be attacked many times with different attack sequence. Based on the efficient computation of the fitness function of an attack, GA can use the information of network performance in the face of different attacks to modify the attack strategy and extract interesting sub-attacks from parents, which is difficult to achieve by other algorithms. Another significant advantage of GA is that it does not depend on a particular network topology. The response of networks with different structures to attacks is different. GA can learn the vulnerability of networks from the previous attack, so as to optimize its attack strategy. GA's search ability mainly depends on the gradient of attack performance near the global or local optimal attack of the network. If the gradient is very large, there is a possibility that GA will miss the optimal solution. But unless a network is deliberately constructed, this situation is very unlikely to happen. So GA has strong applicability to networks of all kinds of topological structure. However, there exists no successful study in the literature for four main reasons. First of all, modeling the network dismantling problem requires a fitness function. In the literature, the quality of an attack is usually measured by the R value Schneider et al. (2011). The R value reflects the change of giant connected component (GCC) size with an ongoing attack to the network; smaller values of R indicate a high effectiveness of the attack. The computation of the R value, however, requires a time complexity of $O(|N| * |E|)$, where $|N|$ is the number of nodes and $|E|$ is the number of edges. This means that even the computation of an attack's quality is very time-consuming for larger networks. Second, the construction of genetic operators, e.g., mutation and crossover is not intuitively clear, compared to other problems modeled as chromosomes. Third, how to choose the initial population is also a challenging problem. The initial population largely determines the search direction and search space of GA, and meanwhile, generating the initial population should not cost too much time. Fourth, GA also involves a parameter optimization issue. Different combinations of parameters can have a huge impact on the results. Some parameter configurations can achieve high solution quality, but correspondingly the runtime can be considerable. Therefore, there is still a trade-off between the quality of the results and the runtime. The major contributions of our study are summarized as follows:

1. We propose an exact fitness function which requires $O(|E|)$ time, compared to the traditional $O(|N| * |E|)$ reported in the literature. Intuitively, our method computes

the R value backward, by reconstructing the graph after the attack took place; which eliminates the need for simulating forward-node removal. This breakthrough made it possible to use GA in network dismantling.
2. We design appropriate genetic operators for the robustness evaluation framework. Specifically, we design crossover and mutation operators based on the notion of local GCC-cascade, which measures the impact of an individual node on the network's functionality in a given attack. By combining the local GCC-cascade information of parents, we are able to create offsprings which combine and preserve the attack-influential nodes of parents. Similarly, we use the local GCC-cascade in the mutation operator to push potentially influential nodes to the front of an attack.
3. We propose several scalable techniques for generating an initial population, based on fast-to-compute centrality metrics as well as a novel condensation-based approximation, which exploit the idea of k-means clustering algorithm and reduces the initial node importance identification problem to significantly smaller network instance.
4. We perform sensitivity analysis on variants of our method, including different choices of the initial population and genetic parameters. Experiments on a wide range of real-world networks reveals that our method significantly outperforms the state-of-the-art, by providing an interesting sweet spot between attack quality and computational complexity.

The remainder of this paper is structured as follows. Section 2 reviews the preliminary concepts of complex networks and how to estimate the network's robustness. Section 3 introduces our framework based on genetic algorithms. The performance of our method against the state-of-the-art is compared in Sect. 4. Finally, Sect. 5 concludes the study and provides some directions for future work.

2 Preliminaries on network robustness

This section introduces relevant background knowledge on the robustness of complex networks. Specifically, Sect. 2.1 revisits fundamental terminology on complex network representation. Section 2.2 reviews the concepts underlying the estimation of network robustness.

2.1 Complex network terminology

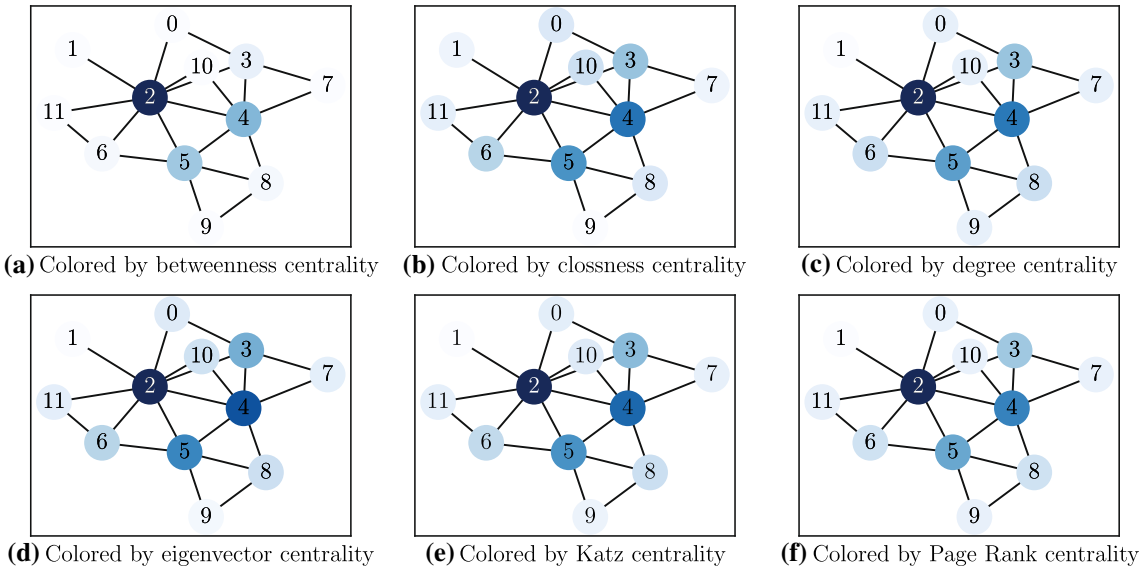
A complex network G consists of a set of nodes N and a set of edges E connecting pairs of nodes. In this study, we restrict ourselves to undirected networks unless otherwise stated; an extension to directed networks is straightforward, with an adjusted definition of robustness. Edges in undirected graphs have no direction, i.e., any two connected nodes can be reached in either direction. Given a network G, one common problem is to measure the importance of nodes in G. This importance can be assessed from different perspectives, depending on the application. Throughout the last decades, several network centrality measures Friedkin (1991) have turned out useful in cross-domain settings; we discuss a few of these common ones below. The simplest node centrality is the degree of a node $n \in N$, which measures the number of neighbors for a given node. This centrality is easy to compute; in fact, it can be read off directly from the adjacency list/matrix representation. Moreover, this metric is attractive, because the intuitive meaning is very straightforward. The degree, however, is a local measure of node importance. It neglects the overall role of a node in the network. Global centralities take a different perspective, by computing a measure on the whole network scale. One example for such a method is node betweenness centrality. Betweenness centrality measures how often a node $n \in N$ lies on the shortest path between all pairs of other nodes Freeman (1977). For an undirected network, betweenness is computed as follows:

$$Betweenness(v) = \frac{2\sum_{(s,t)\in E, s\neq v, t\neq v} \sigma_{st}(v)}{(|N|-1)(|N|-2)} \quad (1)$$

where $\sigma_{st}(v) = 1$ if the shortest path from node s to node t passes through node v (and 0 otherwise). Betweenness centrality is adequate for measuring the importance of nodes assuming a uniform flow between all pairs of nodes in the network. Notably, computing the betweenness centrality of all nodes in the network has a time complexity of $O(|N| * |E|)$. Other common centrality metrics include closeness central-

Table 1 List of abbreviation

Abbreviation	Meaning
GA	Genetic algorithm
GCC	Giant connect component
DEG	An attack method based on degree ranking.
DEGI	An iterative version of DEG.
B	An attack method based on betweenness ranking.
BI	An iterative version of B.
CI	Collective influence.
ND	The network dismantling(also called MinSum) method.
APTA	Articulation point-targeted attack.
AP	Articulation point.
RGB	Residual giant bi-component.
R	Robustness measure R.
$\|N\|$	Number of nodes in the network.
$\|E\|$	Number of edges in the network.
SUS	Stochastic universal sampling.



(a) Colored by betweenness centrality **(b)** Colored by clossness centrality **(c)** Colored by degree centrality

(d) Colored by eigenvector centrality **(e)** Colored by Katz centrality **(f)** Colored by Page Rank centrality

Fig. 1 An illustrative example with a Barabasi–Albert network G_0 with 12 nodes and 20 edges is discussed here. **a–f** visualize the result of six centralities. Darker color represents higher centrality. Although different centralities have a certain consistency, they also make different judgments on some nodes, such as Node 10

ity (measuring the average distance of a node to all other nodes in the network) Sabidussi (1966), eigenvector centrality (the eigenvector of the largest eigenvalue of the adjacency matrix) Bonacich (1972), Page Rank (measuring the cumulative weight of links toward a node) Page et al. (1999), and Katz centrality (a generalization of eigenvector centrality which measures the importance of a node based on the importance of neighbors) Katz (1953).

In order to illustrate the introduced terminology, a small network G_0 with 12 nodes and 20 edges is discussed here. G_0 is an undirected Barabasi–Albert network (n=12, m=2, where n is the number of nodes and m is the number of edges to attach from a new node to existing nodes), which is a scale-free graph with power-law degree distribution. In Fig. 1, we colored the nodes in the network with six different centralities. The darker the color, the higher the centrality is. The importance of nodes estimated by these centralities is mostly consistent with our intuition. For example, Node 1 has the lightest color in almost all charts and, accordingly, if we remove this node, the effects are rather trivial; the remainder of the network will not be affected. On the contrary, if we remove Node 2, the entire network's functionality will be reduced, since Node 2 is in the center and it can be regarded as a hub of the network. Moreover, all centralities rank Node 2, Node 3, Node 4 and Node 5 to be the top four most important nodes, and Node 1 is considered to be the least important node. Different centralities, however, also make distinct judgments about the importance of certain nodes. For instance, Node 10 has a betweenness centrality of 0, since Node 10 only links to Node 2 and Node 4, both of which are directly connected. Therefore, although Node 2 and Node 4 have high betweenness centralities on their own, there is no shortest path going through Node 10. Removing Node 10 will not directly influence the interactions between other nodes in the network. Regarding other centralities, on the other hand, Node 10 still has a rather dark color. This can be understood as one example for node betweenness capturing the role of robustness much better than other centralities.

2.2 Network robustness

The analysis of complex network robustness originates in the area of statistical physics. Specifically, the process of percolation describes the behavior of connected clusters in a random network. Inspired by this concept, the disintegration of a network under a given attack A can be measured as the relative reduction in the size of the giant/largest connected component (GCC), and the GCC is the largest subset of nodes such that there exists a path between all pairs of nodes in the subset. Here, an attack A is a sequence of nodes, i.e., a permutation of N. Smaller sizes of the remaining giant component, after a prefix of A has been executed, can be interpreted as that the attack strategy is more effective. This view follows the intuition that the degree of functionality of a network strongly correlates with the number of connected nodes. In this study, we use the robustness measure R Schneider et al. (2011); an extension to other measures, e.g., effective resistance centrality Clemente and Cornaro (2020), can be considered in future work. The R value directly reflects the change of GCC size along with the attack. In real-world system, the GCC

is the core for the network's ability to function properly. Thus, smaller GCCs lead to poorer network functionality. Note that one is usually interested in the relative size of the GCC compared to the one of the original network. This leads to a normalized GCC size under an attack which has a value between 0.0 and 1.0. In the remainder, we always refer to the normalized GCC size, unless mentioned otherwise. Given a network G and an attack sequence A, the R value for A is defined as:

$$R = \frac{1}{N} \sum_{Q=1}^{N} s(Q) \tag{2}$$

where $s(Q)$ is the size of GCC after removing the first Q nodes in A.

When considering the robustness of a network G, one actually refers to the R value under the most destructive attack. The number of attacks to a network, however, is tremendously large: With $|N|$ nodes, one has $|N|!$ different node sequences. Accordingly, the identification of the best attack and therefore a minimum R is computationally difficult. In fact, it is known that even the underlying node selection problem for a fixed number of nodes is NP-hard, which means that it is unlikely to develop exact techniques even for medium-sized networks. Moreover, it was shown that attempts to model this problem as a mixed-integer problem for solving the problem toward optimality were doomed to fail, given the large search space; a linear solver could not scale beyond a few dozen of nodes. Accordingly, studies in the literature refer to using heuristics for the design of effective targeted attacks. In the following, we review the most recent and frequently used methods for the design of attacks. First, it is easy to see that any node centrality metric can be directly transformed into an attack, by simply ranking all nodes in the network in decreasing order of their importance (assuming that larger centrality values indicate higher importance). For such centrality-based attacking methods, one distinguishes two variants: Static and interactive (or dynamic). In the static case, the centrality values of nodes are computed one time only. This view neglects an important effect under a network attack: Once a node is removed from the network (i.e., it has been attacked), the importance of the remaining nodes often changes. Therefore, the recomputation of centrality metrics after removing a single node from the network, often leads to significantly more effective attacks. Below, we summarize four major centrality-based attacking strategies used in the literature most of which can be traced back to Holme et al. (2002):

- **DEG**: Nodes are attacked in decreasing order of their degree. The degree is computed one time only, making this a static attack. Intuitively, if a node has a high degree, it indicates the node has more (local) influence to the network. Once the node is removed, the direct neighborhood might be either disconnected (in the worst case) or at least the collection/consolidation function, usually performed by hubs in order to reach the economies of scale, will be disrupted. DEG takes only linear time to compute, but it merely focuses on the local influence of a node, which may not work well in some networks with special structures.
- **DEGI**: An iterative version of DEG. Unlike static DEG, DEGI recomputes the node degree each time one node is removed. By recomputation, the procedure for generating a network attack indirectly takes the current attacking prefix into consideration for selecting the next to-be-attacked node Holme et al. (2002).
- **B**: Nodes are attacked in decreasing order of their betweenness value. Betweenness represents the importance of a node at a global level, by considering all shortest paths in a network. Different from DEG, B is able to cut networks into pieces by attacking the center (as induced by the all-pairs-of-nodes flow through the network).
- **BI**: An interactive version of B. Unlike static B, BI recomputes the betweenness centralities each time one node is removed. By recomputation, the procedure for generating a network attack indirectly takes the current attacking prefix into consideration for selecting the next to-be-attacked node Holme et al. (2002).

Another type of methods have gained popularity in recent years, the so-called network dismantling methods. These methods were tailored specifically for generating network attacks. The major motivation for these studies was that degree turned out to be far from the optimal attack; and other centrality metrics were either not significantly better or have a much higher computational complexity. Therefore, these methods were conceived with the goal to compute attacks efficiently even for large networks. The concepts underlying these methods vary; we summarize these methods below:

- **CI**: The collective influence (CI) of a node is an index of node importance based on percolation theory Morone and Makse (2015). Technically, it can be understood as a kind of extension to the degree centrality to a higher dimension. The CI value of a node is defined as follows:

$$CI_l(i) = (k_i - 1) \sum_{j \in \partial B(i,l)} (k_j - 1) \tag{3}$$

where k_i is the degree of node i, $\partial B(i, l)$ is the frontier of the ball of radius l centered on node i. In each iteration, this algorithm removes the node with the highest CI value, until the entire network is dismantled. The advantage of

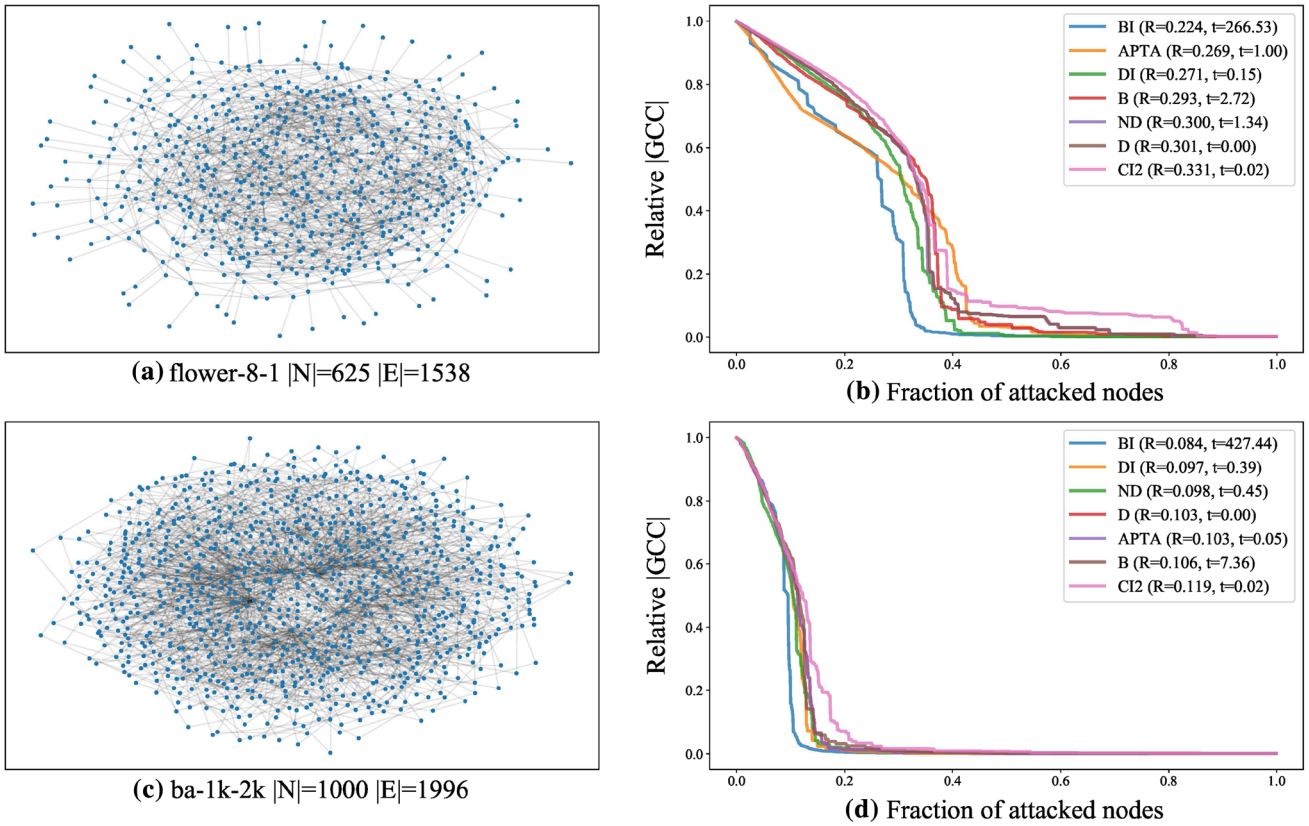


Fig. 2 Two example networks (left) and attack curves visualizing the change in GCC size with an ongoing attack (right). In the legends, all seven methods are ordered by increasing value of R. It is easy to see that BI always outperforms other methods, but it also takes the longest runtime. The differences between different attack strategies are quite considerable. For the flower-8-1 network, the range of R is between 0.224 and 0.331, which indicates that the current heuristic methods still have a large space for improvement

CI is its accuracy of node importance evaluation under $O(|N|*log|N|)$ time complexity for strictly hierarchical networks. Given its semi-global design, as controlled by the ball size l, CI can identify nodes with low degrees but with important roles in network structure and function.

- **ND**: The network dismantling (also called MinSum) algorithm is designed around a three-stage procedure Braunstein et al. (2016). At first, the network is being decycled. Intuitively, a network without cycles is much easier to be attacked, following the intuition of CI. As a second step, the tree-like network is broken into further pieces, until an externally set size threshold C is reached. Finally, a few short cycles are re-introduced to the network, by improving the attack further, while maintaining the maximum size under the user-set threshold.
- **APTA**: Articulation point-targeted attack (APTA) Tian et al. (2017) exploits the critical role of articulation points (APs) in a network. An AP is a node in the network whose removal increases the number of connected component. A depth-first search-based algorithm developed by Tarjan can identify the articulation points in linear timeTarjan and Vishkin (1985). Moreover, with some book-keeping during the traversals, it can also acquire the information of component sizes after removing each AP, without the need for an explicit simulation. Note that removal of an AP may result in the creation of new APs and the disappearance of the old APs. APTA iteratively removes the most destructive AP whose removal results in the smallest GCC size. After removing all APs, the leftover is called residual giant bi-component (RGB) in which there are at least two different paths between any two nodes. The RGB can be further dismantled through removing the nodes with the highest degree. The worst case time complexity of APTA is $O(|E|*log|N|)$, making APTA scale up well with the size of the network.

In order to have an intuitive understanding of these methods, we choose two networks with different topological structures and show the results of attack strategies in Fig. 2. The robustness curves on the right-hand side show how the GCC size changes as the number of removed nodes increases, the performance of different methods can be quite different.

We can see that BI always obtains the lowest R value (i.e., best attack sequence), although these two networks have totally different topological structures. In fact, BI has a rather strong generality, it performs excellently on most networks. Other attack strategies are often limited to certain networks Wandelt et al. (2018).

In summary, all these methods have in common that they do not compete well even against interactive betweenness, let alone yielding close-to-optimal results. Their contribution lies in the ability to compute a ranking on the network, which usually outperforms the simple attacks based on degree. In terms of the generality of the algorithm, some perform well in networks with a particular topology (e.g., CI works excellent on hierarchical networks), but it has shortcomings when dealing with other, real-world networks. On the other hand, for networks without (critical) APs, APTA cannot perform well by design. Overall, these methods indeed have shorter runtime than BI, but they are inferior in terms of the result quality, and are far from achieving a high quality attack.

3 Solution methodology

In this section, we introduce the specific design and details of our algorithm. Section 3.1 introduces the overall structure of our algorithm. Section 3.2 introduces the chromosome representation of network attacks. Section 3.3 presents the fitness function as well as a novel linear-time algorithm for computation of the R value. Section 3.4 reports our strategies for deriving the individuals in the initial population. Section 3.5 develops the genetic operators, covering selection, crossover, and mutation.

3.1 Overall algorithm

We exploit GA for computing high-quality attacks in order to address the network dismantling problem. GA starts from an initial population, which consists of a collection of individuals. Each individual in the population represents an attack to the network G under consideration. We have designed three initial population generation methods. These three initial population generators have different focuses and applicable network types. Their combination can provide a potential initial population for GA. Throughout the process of evolution, GA generate succeeding populations by the means of selection, crossover, and mutation. In crossover, we hope that the good genes of parents will be passed on to their children. Here, we use the size of GCC after the node is removed to determine the priority of the node. If the size of GCC of the network after the node is removed is large, it indicates that the node is in the front of the attack sequence, and the probability of the node appearing in the front of the attack sequence among the child nodes is high. We find that this crossover approach can effectively improve the attack quality, that is, reduce the R value. In mutation operators, we want to place nodes that have a critical impact on network connections at the front of the attack sequence. Here, we choose the change in GCC size to measure the importance of the node. If the GCC size of the network is significantly reduced after removing a node, the node should be attacked earlier, hoping to move the cascade forward. In order to perform an informed evolution, a fitness function needs to assess the quality of an individual in the current population. In our case, the fitness function measures the quality of an attack to G. Since the fitness of an individual is reciprocal to the R value, we let $f = \frac{1}{R}$. We have proposed an efficient algorithm for computing R. Although R has been proposed for a long time, all current methods of calculating R require quadratic runtime. We reduce the runtime to linear by the inverse process of the attack, namely the process of rebuilding the network. Individuals with higher fitness values (accordingly smaller R values) have a greater likelihood of being chosen and retained in the next generation. After reaching a specific termination criterion, the best individual is taken as the solution obtained by GA. The flowchart of the algorithm is shown in Fig. 3. Notably, the solution is not guaranteed to be optimal, since GA can get stuck in local minima or specific parts of the search space, but we show that GA exhibits excellent performance (see Sect. 4).

3.2 Chromosome representation

Finding the best attack to a network G is, in essence, a node ordering problem: We aim to find the sequence A of nodes, i.e., a permutation of N, such that the R value of A with respect to G is minimal. The representation of an attack is a chromosome consisting of $|N|$ distinct node identifiers. The first node in the chromosome is attacked first, and the last node is attacked last; nodes in intermediate positions are attacked accordingly.

3.3 Fitness function

At the heart of GA, we need a fitness function to evaluate the quality of individuals. For network dismantling, the concern is the giant connected component (GCC) Kitsak et al. (2018) size after removing a fraction of nodes. Therefore, we use the R value to evaluate the quality of a chromosome. The standard way for computing the R value is to simulate the node removal as induced by the attack A. During the process of simulation, the (relative) size of the GCC is recorded at each step until the entire network is dismantled. While this method is easy to implement, it has an obvious pitfall. The runtime complexity is $O(|N|*|E|)$. The reason is that this process requires the execution of depth/breadth-first search on the whole network for each iteration, which means

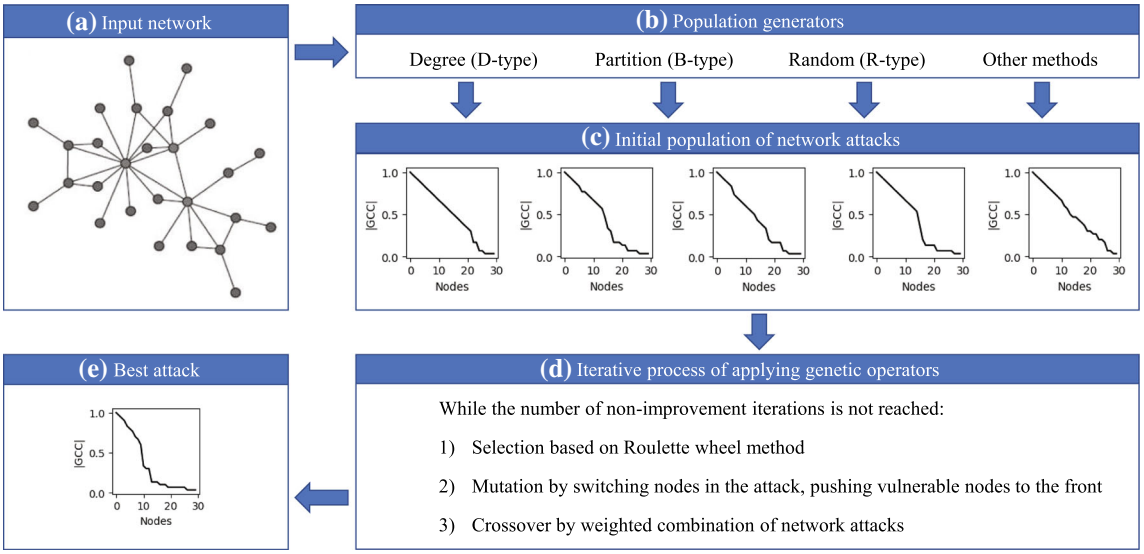


Fig. 3 GA starts from generating the initial population, then the population will be selected, mutated and crossovered through an iterative process, until the termination criterion is met. At last, the best individual will be obtained as the result of the whole algorithm

a single traversing of all edges, i.e., $O(|E|)$ is needed in the worst case. Since there are N iterations, the overall complexity is $O(|N| * |E|)$. Accordingly, the computation of R values for larger networks is intractable. While the computation can be accelerated by sampling Wandelt et al. (2017), the accuracy of the results deteriorates significantly and unpredictably. Accordingly, it is difficult to design a GA as long as the runtime of the fitness function is at least quadratic in the size of the network or the sampling is unacceptably high.

The process of R value computation can be considered as a forward calculation (assuming an attack traversal from left to right). In this study, we propose to compute the R backward. Instead of attacking the network in order of the attack sequence A, left to right, we build up the network in the reverse order of A. The detailed process is shown in Algorithm 1. We start with a set of $|N|$ disconnected components, each containing exactly one node. Then, we traverse the attack from right to left. During this process, we recover all edges which must be present in the network at that stage of dismantling. In order to compute the GCC size efficiently, we use the union-find data structure Conchon and Filliâtre (2007). Each union-find tree represents one connected component. When we start with the last node of the attack sequence, we add its links to the right components and merge the corresponding union-find trees of the two edge's nodes. This process is repeated for $|N|$ times, until the entire network is recovered. In each iteration, the size of GCC is simply the size of largest union-find tree. Therefore, we can keep track of GCC size while building up the network efficiently. Figure 4 shows the process of this algorithm on a small network G_0. The time complexity of this improved R computation is $O(|E| * log|N|)$, compared to $O(|N| * |E|)$ for the traditional variant. The time complexity is obtained since we rebuilt the network edge by edge and adding one edge takes $O(log|N|)$ steps for merging the union-find trees.

Algorithm 1 Efficient computation of R (pseudo code)

Input: Network G with nodes N and edges E, attack sequence A
Output: R

```
isbuilt ← {n : False}  ∀n ∈ G
A ← A.reverse()
UF ← UnionFind(G)
for node ∈ A do
    isbuilt[node] ← False
    for neighbor ∈ G.neighbor(node) do
        if isbuilt[neighbor] then
            UF.Union(node, neighbor)
        end if
    end for
    GCCs.append(UF.largest/N)
end for
return sum(GCCs)/|GCCs|
```

3.4 Initial population

Genetic algorithms start with an initial population, which provides an initial pool of genetic information. While the fitness of the initial population might be very bad, genetic operators (see Sect. 3.5 below) aim to gradually improve the fitness. One obvious way for generating an initial population is to simply generate random individuals, which corresponds to random attacks, intuitively. Random attacks, however, are usually very inefficient in dismantling real-world networks, given networks' ability to withstand such attacks. Therefore, it is beneficial to provide non-random individuals to the initial gene pool. One critical factor for the initialization is that

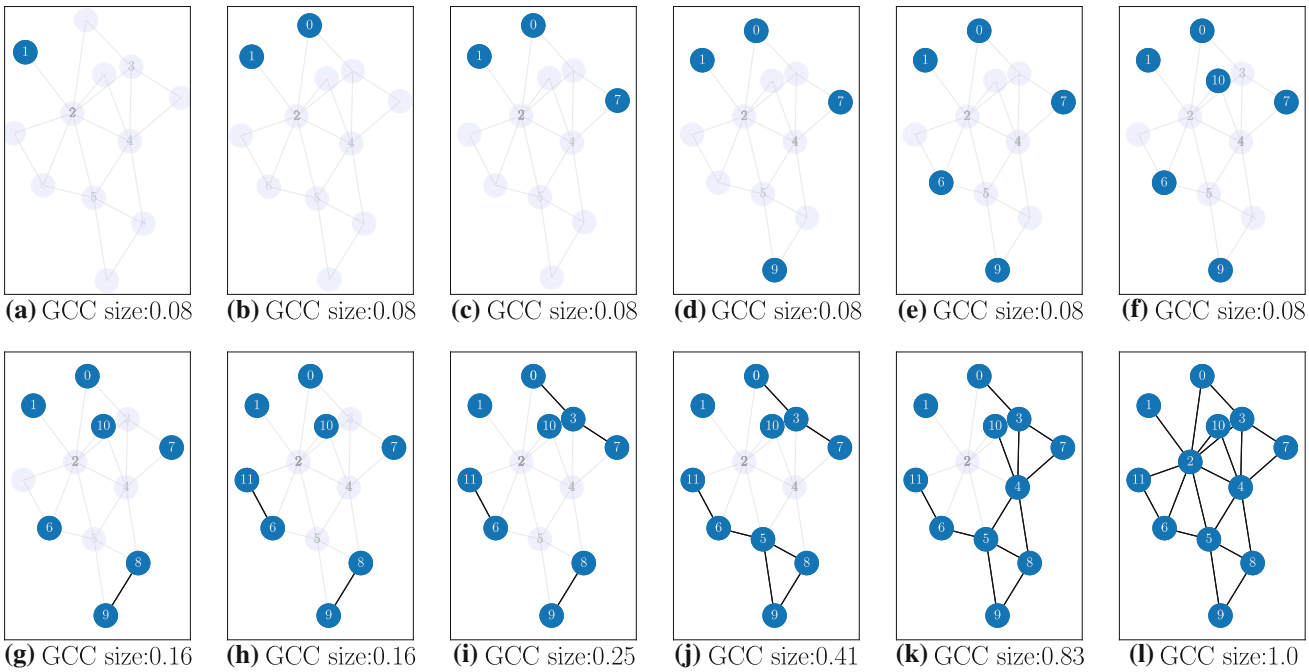


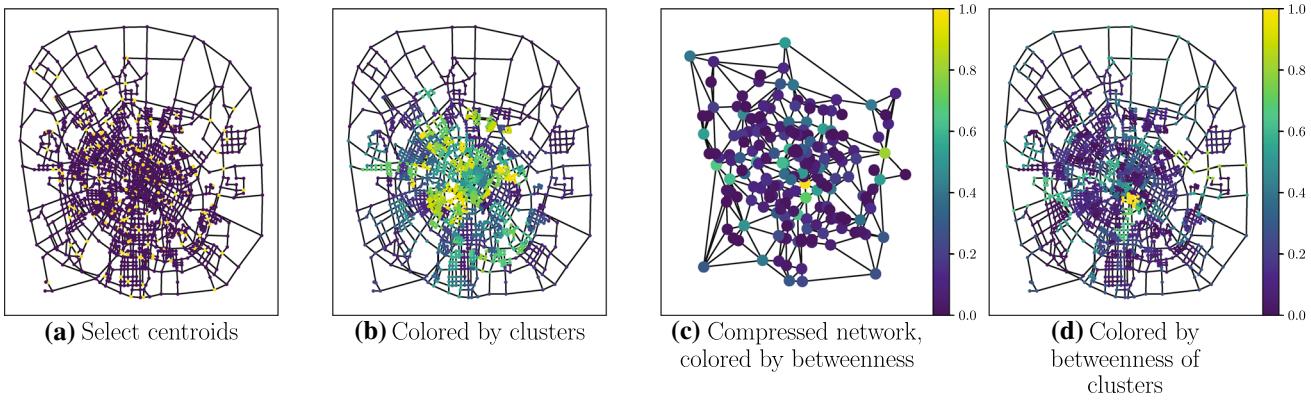
Fig. 4 Process of efficient computation of R value on G_0. We compute the R value in a backward way. Starting from N disconnected components, we traverse the attack sequence backwards and recover the corresponding edges and nodes during this process. The union-find data structure is used to keep track of these connected components, thus it is easy to compute the GCC size in each step. The time complexity of this improved R computation is $O(|E| * log|N|)$

the attacks can be generated fast. We discuss our initialization methods below. In total, there are three types of initial individuals in our study:

1. **D**-type: This type of individual is obtained by computing the result of DEG attacking strategy. DEG can be computed very fast (linear time). Having a degree-based individual in the initial population can make the offspring have the potential to put the nodes with high degree centrality in the front of the attack sequence, which may improve the quality of the attack. In our experiments, we found that DEG often provides a good starting point for GA. Only in case of narrow degree distributions, i.e., many nodes have highly similar degree values, degree is not useful as an informed initialization method.
2. **B**-type: This type of individual aims at giving hints on the potential betweenness centrality of nodes. Since computing the actual betweenness centrality is computationally too expensive, we propose a kind of approximation here. We compress the network and execute the betweenness computation on this compressed (and much smaller) network. Accordingly, the problem becomes how to partition the network in a fast way, such that, each part is compressed into a single node. Here, we exploit the idea of k-means clustering algorithm Alsabti et al. (1997), by first randomly selecting $\sqrt{|N|}$ nodes as centroids and then assigning the remaining nodes to the nearest centroid through a multiple starting points breadth-first search process. Each centroid and the nodes assigned to it are regarded as a cluster node. In the compressed network, clusters are contracted into single nodes, and the links between different clusters are edges. After that, we compute the betweenness centrality in this compressed network. Finally, for generating the initial individuals, the nodes are sorted according to the betweenness of the cluster they belong to and the degree of the nodes. Figure 5 visualizes the process of this algorithm. The naive partition algorithm performs rather well in our experiments.
3. **R**-type: This type of individual is generated randomly. It is simply a random permutation of all nodes and can be computed in linear time. The result of random attack may be poor, but random individuals could provide more directions for population evolution. We believe random individuals can help to discover interesting parts of the search space, which are not (directly) covered by centrality metrics.

3.5 Genetic operators

The operators covered in standard genetic algorithms include selection, crossover, and mutation. We describe below how



(a) Select centroids **(b)** Colored by clusters **(c)** Compressed network, colored by betweenness **(d)** Colored by betweenness of clusters

Fig. 5 Example visualization of the B-type individual generation for the road network of Chengdu with 1,902 nodes and 3,052 edges. **a** Randomly select $\sqrt{|N|}$ nodes as centroids. **b** Assign the remaining nodes to the nearest centroid, forming $\sqrt{|N|}$ clusters. This can be done by a multiple starting points breadth-first search process. **c** Compress each cluster to one node in the compressed network, and compute the betweenness centrality. **d** To generate the B-type individual, nodes in the original network are sorted according to the betweenness of the cluster they belong to

these three operators are implemented for the network dismantling problem.

3.5.1 Selection

The first operator is the selection of individuals for mating. The goal is to preferably select individuals with a high fitness. We apply fitness proportionate selection with the stochastic universal sampling (SUS) Baker (1987). Unlike fitness proportionate selection, SUS uses a single random value to select all individuals by choosing with evenly space interval, which give individuals with small fitness value a chance to be chosen. In addition, for the sake of avoiding population degradation over time, we implement elitism, i.e., we always retain the top ten percent individuals to the next generation Ahn and Ramakrishna (2003).

3.5.2 Crossover

After the parents are selected, we need to compute the offsprings, based on the parent chromosomes. The standard procedure for performing the crossover is to split the parent chromosomes at one or more places and then distribute the segments to the children. This naive crossover does not work here, since we have to make sure that the attacks in each child cover all nodes in the network. Instead, all nodes in parent chromosomes are weighted by a random number between zero and the absolute size of GCC after removing the node. Thus, the nodes in the front are more likely to have higher weights. Then, the nodes are sorted by their weights and the result of sorting is the offspring chromosome. The pseudo-code for crossover is shown in Algorithm 2.

Algorithm 2 Crossover operator (pseudo code)

Input: $parent_A, parent_B, GccSize_A, GccSize_B, networkG$
Output: $child_A, child_B$
1: $Ave\_GccSize \leftarrow \{n : (GccSize_A[n] + GccSize_B[n])/2\}\ \forall n \in G$

2: $weights_A \leftarrow \{n : random.uniform(0, Ave\_GccSize[n])\}\ \forall n in G$
3: $weights_B \leftarrow \{n : random.uniform(0, Ave\_GccSize[n])\}\ \forall n \in G$

4: $child_A \leftarrow sorted\ n\ by\ weights_A[n]$
5: $child_B \leftarrow sorted\ n\ by\ weights_B[n]$
6: **return** $child_A, child_B$

3.5.3 Mutation

The mutation operator maintains the diversity among chromosomes and is able to introduce previously unseen properties Smith and Fogarty (1996). For a given attack sequence, we cannot simply change individual nodes, because we need to cover all nodes in an attack. Accordingly, the idea is to select two nodes and exchange their positions. In order to reduce the R value, our goal is to move more destructive nodes to the front of the attack sequence. We quantify the importance of a node i with a special value, Δ^i_{gcc}, which is defined as the change of GCC size after removing the node i. This Δ_{gcc} is obtained while computing the fitness value. We select two nodes, one with high Δ_{gcc} and the other with low Δ_{gcc}, still using the roulette wheel method. All the nodes are weighted by Δ_{gcc} when selecting the first node and $1 - \Delta_{gcc}$ when selecting the second node. Then, we switch the one with higher Δ_{gcc} to the front, with the intention that the switch will make the network cascade appear earlier. This process is repeated for $\frac{|N|}{2}$ times for each chromosome (See Sect. 4.1 for sensitivity analysis of GA). However, in our experiment, we found that the effect of mutation became more negative, i.e., mutation are more likely to worsen the

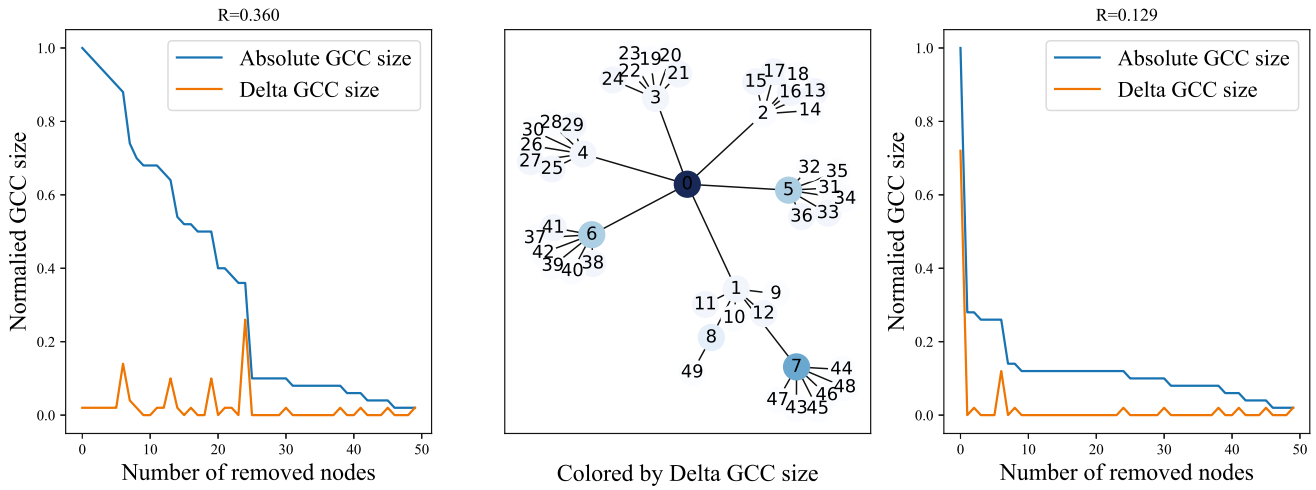


Fig. 6 Process of mutation. **a** The attack curve of an individual is shown. The orange line represents the change of GCC size after removing one node. It is easy to see that the node with highest Δ_{gcc} is in the middle of the attack sequence. **b** The network is visualized here, nodes are colored by their Δ_{gcc} value. Node 0 has the darkest color, i.e., highest Δ_{gcc} value. **c** After we switch Node 0 to the front of the attack sequence, the network cascade appears earlier, resulting in much lower R value

individual, with an increasing number of iterations. Therefore, we set a parameter mutation decay D to control the mutation probability of an individual. The mutation probability is $P_{mutation} = \frac{1}{n^D}$, where n is the number of iterations. (See Sect. 4.1 for more detail about the specific parameter adjustment.) Figure 6 visualizes the process of one mutation. Node 0 has the highest Δ_{gcc} value, but it is not in the front of the original attack sequence. We can see that after switching node 0 and node 21, the R value of this attack sequence is reduced significantly.

4 Experimental evaluation

In the following section, we evaluate all techniques on a range of real-world networks. These networks were obtained from network repository Rossi and Ahmed (2015), the largest, interactive network data repository. All network data is provided in a standardized format. The networks come from a variety of applications and domains (e.g., network science, bioinformatics, machine learning, data mining, physics, and social science). In total, 49 networks were selected for our study. The number of nodes in these networks is 544–4,182 (with an average of 1,353), and the number of edges is 1,538–4,284 (with an average of 2,994). Overall, this makes the networks in our study rather sparse and fragile, which makes sense intuitively, since we are interested in computing harmful network attacks.

The remainder of this section is organized as follows. First, we report the results of sensitivity analysis regarding the parameters of our GA method (Sect. 4.1). Given that GA is a non-deterministic algorithm (unless a fixed seed is used), we report on the variance of results under different seeds in Sect. 4.2 and the impact of initial solution generators in Sect. 4.3. In Sect. 4.4, we report on the comparison of GA with the state-of-the-art in the dismantling literature. In addition, Sect. 4.5 assesses the ability of GA to improve a given network attack, as computed by the state-of-the-art. Section 4.6 investigates a different perspective, exploring whether we can achieve results on par with BI, without actually using BI. Finally, Sect. 4.7 reports further results on the scalability of GA.

4.1 Sensitivity analysis of GA parameters

In a first set of preliminary experiments, we perform sensitivity analysis on five selected parameters of GA. These parameters are:

- **Population size**: The number of individuals in the population.
- **Non-improvement iterations**: The termination criterion in our implementation, defining the maximum number of iterations without improvement.
- **Elitism rate**: The ratio of the best individuals which are preserved from one generation to the next one.
- **Mutation rate**: The number of genes (nodes) which are mutated for a new offspring.
- **Mutation decay**: This number controls the mutation probability of an individual and how the probability decreases in later generations.

We have performed sensitivity analysis on seven networks. These seven networks come from different domains and have

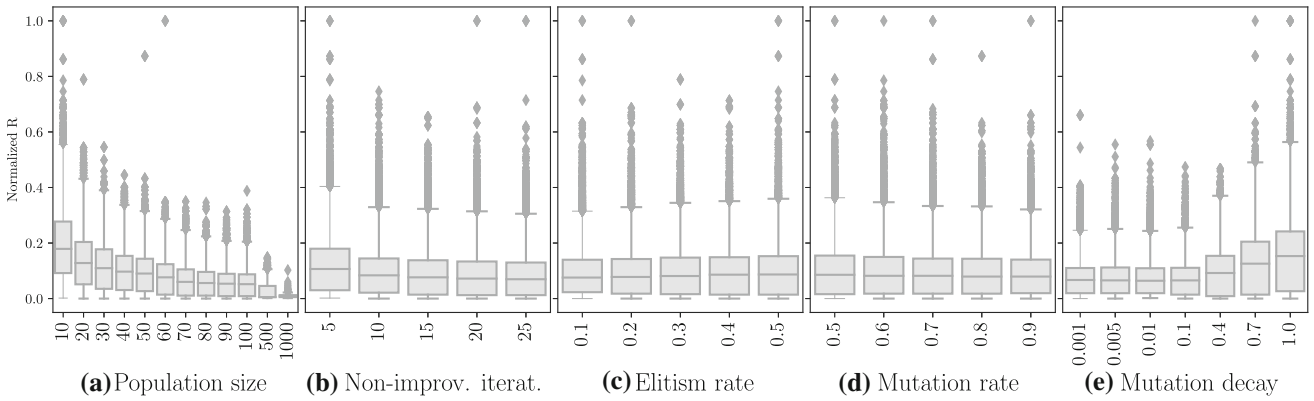


Fig. 7 Analysis of genetic algorithm parameters in terms of the quality of network attacks

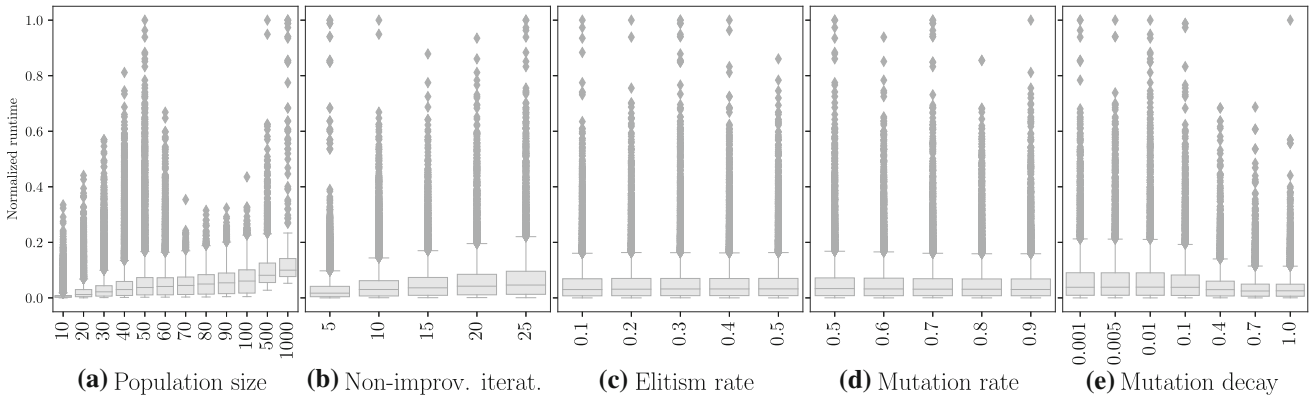


Fig. 8 Analysis of genetic algorithm parameters in terms of runtime

different topologies. We mainly analyze the results of our algorithm under different parameters from two dimensions: Attack quality and runtime. Here, in order to show the comparison results straightly, we have normalized both the R values and the runtime with the minimum value to 0, and the maximum value to 1. Figure 7 and Fig. 8 show the results of this experiment. For population size, with the increment in individuals, the runtime also increases significantly. It is almost a linear relationship. At the same time, the R value has also gone down. We can observe that as the population size increases, the decreasing trend of R value gradually flattens out, which shows a marginal effect. The increase in non-improvement iterations leads to the increase in runtime, but surprisingly, there was no significant improvement in the quality of the results. Elitism rate and mutation rate have no influence on both runtime and attack quality. As for mutation decay, larger mutation decay will cause worse results but with shorter runtime.

According to the analysis above, we select three representative parameter settings, which have different performance in terms of the speed and quality of the results, respectively. GAF is more focused on runtime, GAH is more focused on quality of results, and GAM is an intermediate version. The details are shown in Table 2. In order to reduce the variables, in the rest of the study, we use these three GA instances for specific experiments.

4.2 Spread/distribution of R-values and runtime for different runs of GA

GA is a non-deterministic algorithm, i.e., even for the same input, GA can exhibit different behaviors on different runs. Hence, we should further check the stability of GA. We carried out the experiments on 49 medium-size networks. Each GA instance attacked the same network for seven times. For each GA instance and each network, we compute the standard deviation of the runtime and R values. The results are shown in Fig. 9, where darker part in the lower left corner indicates more stability. In the figure, we can clearly observe that the three GA instances have different stability for runtime and R value. GAF has a narrow runtime distribution, but the standard deviation of its R value is the highest. GAH shows the opposite, it has a narrow R value distribution but it is not stable regarding runtime performance. GAM maintains a good trade-off between the stability of runtime and quality of results.

Table 2 Three GA instances of different parameter settings. GAF focuses more on runtime; GAH focuses more on the quality of the result; GAM is a trade-off between both

GA instances	Population size	Non-Improv. Iterat.	Elitism rate	Mutation rate	Mutation decay
GAF (Fast)	10	5	0.5	0.5	1
GAM (Median quality)	30	10	0.5	0.5	0.4
GAH (High quality)	50	15	0.5	0.5	0.1

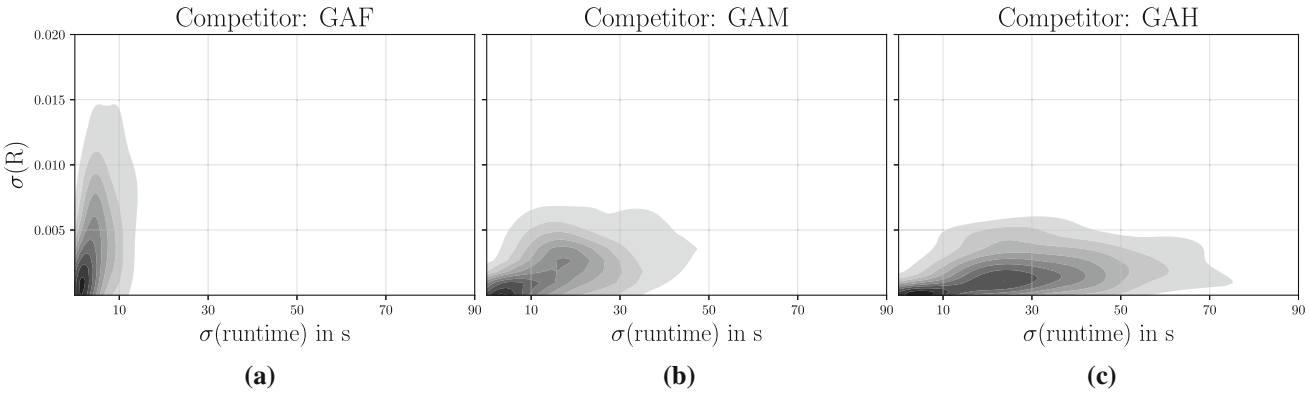


Fig. 9 Standard deviation of the runtime and R value of three GA instances

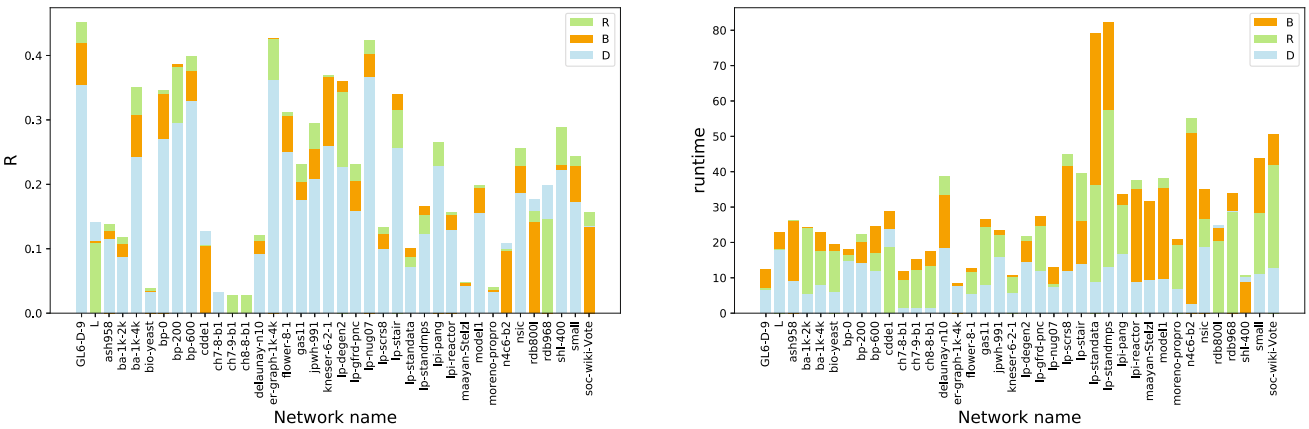


Fig. 10 Comparison for B/R/D-type individuals. We did experiments on 38 networks. It can be clearly seen that D-type is superior to the other two both in terms of solution quality and running time. In most networks, R-type has the worst performance and takes longer to run than D-type. B-type's attack quality is in the middle, but it takes the longest running time

4.3 Comparison for initial population generators

In Sect. 3.4, we introduced three initial population generators. In order to compare the effects of different initial populations, we conducted comparative experiments on 38 networks. The result is shown in Fig. 10. In terms of the quality of results, D-type has an obvious advantage since it can obtain the attack sequence with the lowest R value in most networks. R-type has the worst attack quality. This is intuitive, since random attacks often do not hit the network's weaknesses very well. The result quality of B-type was slightly better than that of R-type, but there was still a gap compared with that of D-type. In terms of running time, D-type still performs best, even faster than the purely random R-type. The B-type takes the longest running time. However, in the actual application of GA, compared with only using D-type, we prefer that the above three initial population generation methods should be mixed. Although D-type performs best, in some networks with narrow degree distribution, D-type, based on degree centrality, will lose its advantage. Accordingly, we recommend having at least one D individual and at least one P individual; the remaining individuals can be filled up with R individuals.

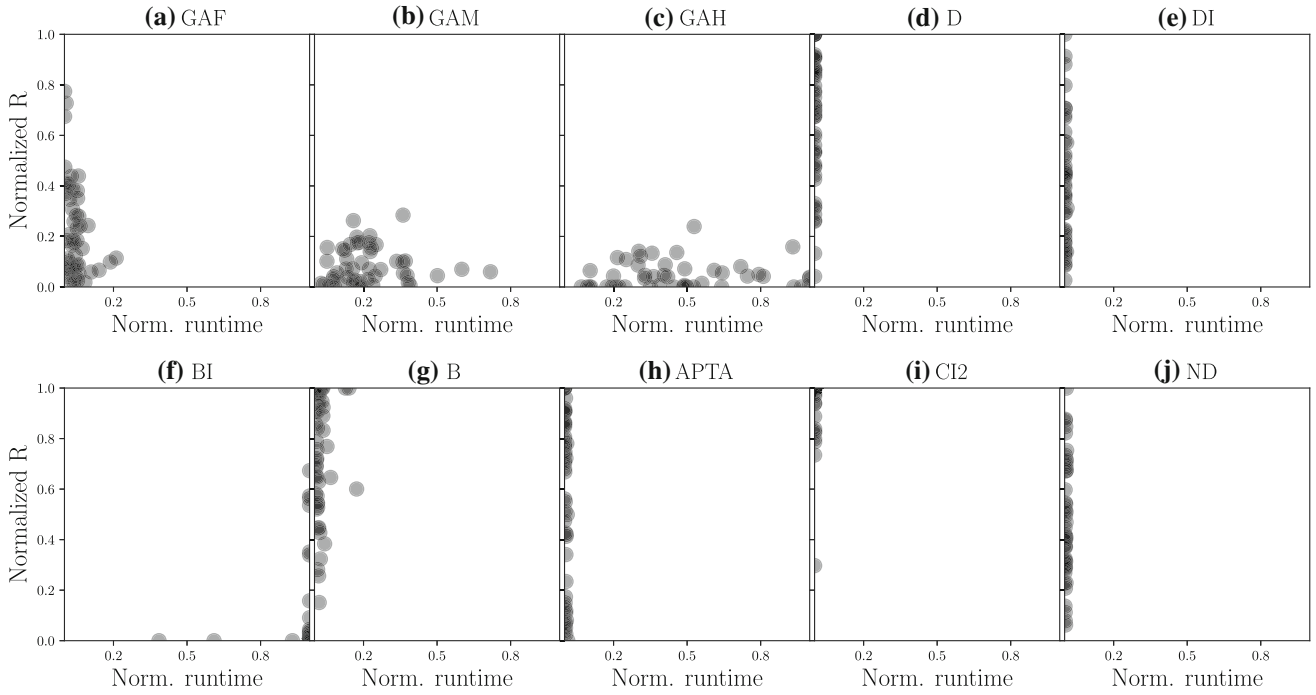


Fig. 11 Overall comparison between GA and other methods. Each subgraph contains 49 circles representing the results of attacks on 49 networks. To avoid contingency, we repeatedly attack the each network with GA instances for seven times and select the median of the results to represent the performance of GA instances. It can be seen that GA can get rather low R values while spending much shorter runtime than BI

4.4 Overall comparison of GA against other methods

In Sect. 2.2, we describe in detail other representative methods that are commonly used in the network dismantling literature. In a comparative analysis of approaches to network dismantling, the authors identified betweenness interactive (BI) as a reference competitor in terms of quality, but at the cost of a quadratic cubic increase in running time Wandelt et al. (2018). Therefore, we use this method as a baseline. In this subsection, we make a comparison of GA and other methods. The results are shown in Fig. 11, and we also normalized the runtime and R value. Each subfigure has 49 circles, respectively, representing the attack results of the corresponding algorithm on a network. Considering that GA is a non-deterministic algorithm, in order to avoid contingency, we use GA to attack each network for seven times and use the median of the results to represent GA's performance to this network. As can be seen in the figure, BI can get the best result in most networks, but it also takes the longest time, its data points are concentrated in the lower right corner. The scattered points in the GA's chart are distributed along the x-axis, which means that GA can get the results with rather high quality, while not spending such long runtime as BI. The other six methods can be executed in short time, but the quality of results is worse. In summary, GA significantly outperforms the state-of-the-art methods. It can get results very close to BI while spending much less runtime.

In order to more accurately illustrate the results of GA comparison with other methods, we used the Bayesian signed-up ranks test to compare GA with other competitors Demšar (2006)Benavoli et al. (2017). This method was originally used to compare classifiers in machine learning. By changing the classifier to the network dismantling method and the data set to the network, we can apply this Bayesian comparison to the comparison of the network dismantling method. We set the region of practical equivalence (rope) to 0.01, meaning that if the difference of R between the two methods is less than 0.01, we assume that the two algorithms behave similarly. The results are shown in Table 3. It can be seen that GA can win overwhelmingly for all other algorithms except BI. Even when compared to BI, GA can maintain similar performance in high probability. Considering that GA is much faster than BI in runtime, we think that GA achieves a good tradeoff between result quality and runtime.

To verify the wide applicability of GA, in addition to R, we also used another measure to compare GA with other methods. The network functions through its GCC, so removing a single node has a limited impact on network integrity. However, by removing multiple nodes from a network, the network can be broken up into independent components. An

Table 3 Bayesian signed-up ranks test between GA and other algorithms. We used the Bayesian signed-up ranks test to compare GA with other competitors. The table shows the probability that GA can get better results than other methods. For BI, GA has a high probability to obtain results with similar attack quality. For other algorithms, GA can always get results with lower R values

Competitors	D	DI	BI	B	APTA	CI2	ND
Win	1	0.994	0	1	1	1	1
Lose	0	0	0.304	0	0	0	0
Practical equivalence	0	0.006	0.696	0	0	0	0

interesting question is how many nodes must be removed before the network can be broken down into independent components thus lose its function. Another commonly used measure, which we will call C here, is defined as the percentage of nodes to be removed which make the GCC size less than 10% of the total number of nodes in the network. We compared the attack results of the three GA instances with other methods based on this metric, and normalized the results. As shown in Fig. 12, BI still performs best, usually with the fewest nodes removed. The distribution of GA is also concentrated on smaller C, indicating that GA is significantly better than other algorithms except BI. We can see that GA can also achieve performance close to BI under different criteria, which illustrate the generality of GA based on R value.

4.5 How does GA improve the results of other methods?

GA can be used not only as an attack strategy itself, but also in combination with other methods to further improve their results. Here, we present the results of seven existing methods to the initial population of GA, respectively. We compute the R improvement of different GA instances. The R improvement is normalized by using $Normalized\ R_{improvement} = (R_{original} - R_{final})/R_{original}$. The increased runtime is also normalized in a similar way. The results are shown in Fig. 13. For different conventional methods, the benefit of GA varies. It depends on the quality of the method itself. GA can reduce the R value by $20\% \sim 40\%$. As for the runtime, many of the conventional methods we selected here have linear time complexity, so GA's runtime increases considerably relative to their own runtime. Different GA instances vary in their performance in terms of increased runtime, with GAH being several times as large as GAF. However, for methods with quadratic or higher time complexity, the additional runtime by GA is trivial. In combination with the result quality and the runtime, GA can provide a good trade-off to significantly improve the quality of the results by increasing the runtime within a reasonable range.

4.6 Can GA beat BI, without using BI?

By far, BI is the best algorithm if only considering the quality of the results. Compared with other state-of-the-art methods,

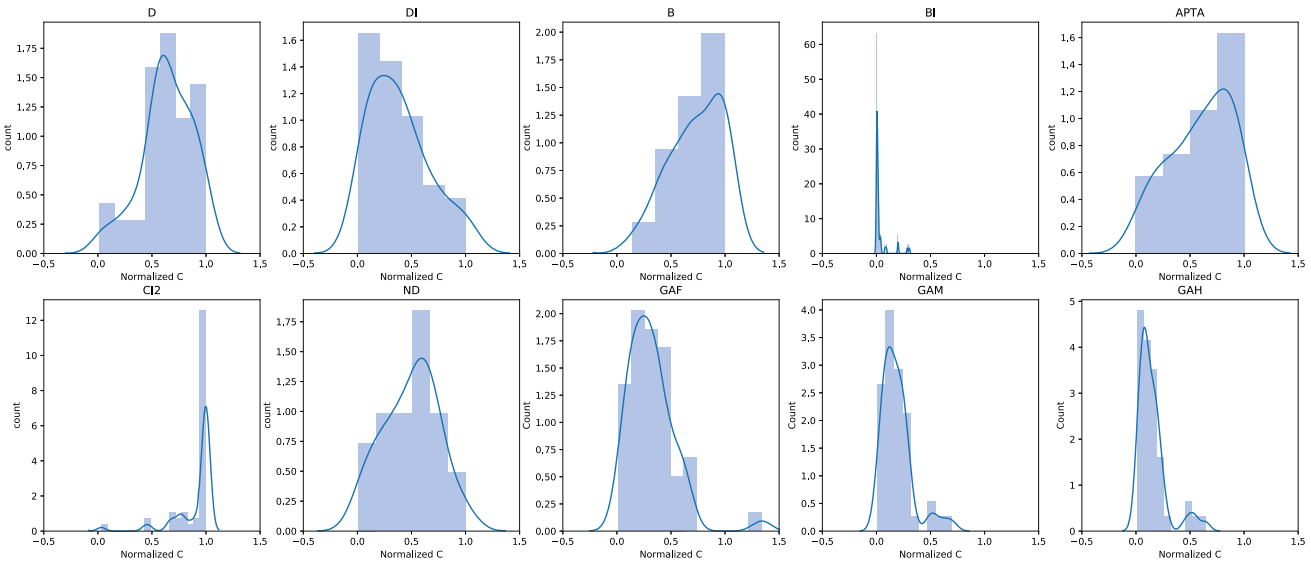
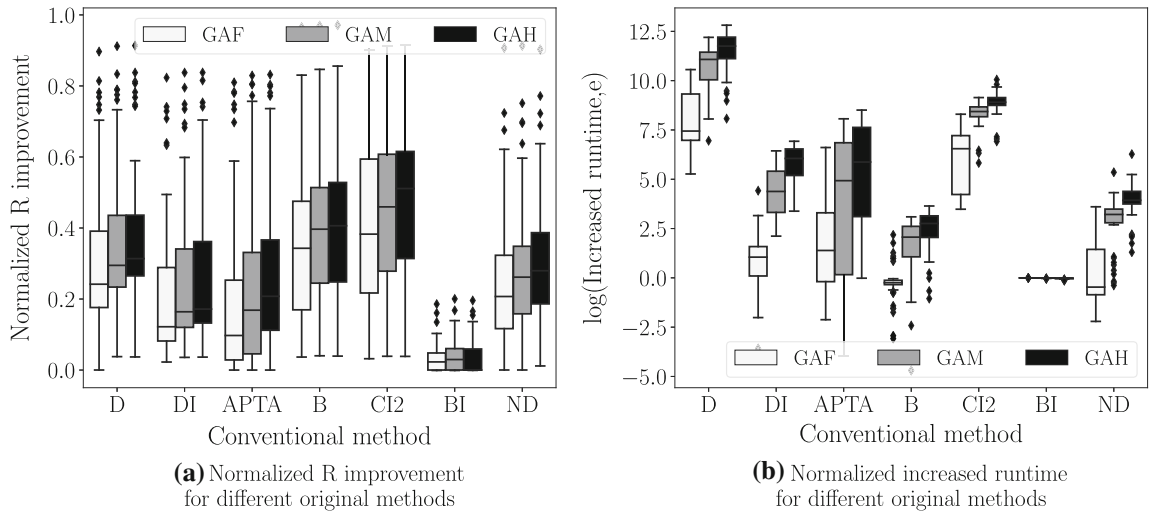


Fig. 12 Comparison for different network dismantling algorithms based on metric C. We compare the attack results of seven common network dismantling algorithms and three GA instances. The graph shows the quality of the attack results by these ten competitors on 49 networks. We normalized the results, mapping them linearly on the interval from 0 to 1. BI still gets the best results. With the exception of BI, GA can outperform other competitors by a wide margin



(a) Normalized R improvement for different original methods

(b) Normalized increased runtime for different original methods

Fig. 13 Improvement GA makes based on different conventional methods

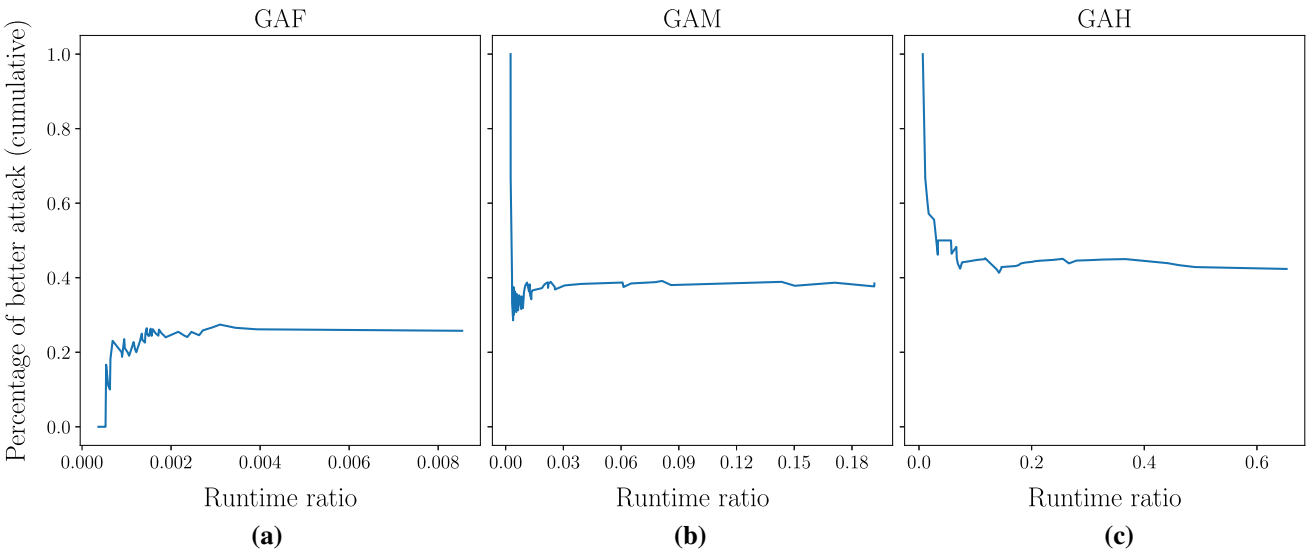


Fig. 14 GA based on other conventional methods competes with BI. The x-axis value is the ratio of GA runtime to BI's. The y-axis value represents the ratio of GA beating BI when the runtime ratio is less than the corresponding value of x-axis

it can get the best results on most networks. However, BI takes cubic runtime, which makes it difficult to scale up to larger networks. Here, we make a comparison between BI and GA with different initial populations, and see to what extent GA can compete with BI. Figure 14 shows the results of three GA instances. The x-axis is the runtime ratio of GA and BI. The y-axis represents the ratio that GA results are better than BI when the runtime ratio is less than the corresponding value of x-axis. As a result, GA takes far less time than BI. Even the slowest GAH is mostly less than half the runtime of BI, while for the faster GAF only takes less than 1% runtime. In terms of the quality of the results, although only a small part of the results can exceed BI, the difference between the results of most GA and BI is rather minor. The quality of the results slightly deteriorated, while the runtime is greatly improved. This means that GA can be applied to large-scale networks, making it more practical in real-world applications.

4.7 Runtime analysis of GA

GA does not come with a fixed time complexity that can be calculated analytically like other methods. The runtime of GA can be considered as the product of average number of iterations and the runtime of each iteration. We found in the experiment that the runtime of each iteration was mainly spent on the fitness function. As we have discussed in Sect. 3.3, the time complexity of computing the fitness function is $O(|E| * log|N|)$, where $|E|$ represents the number of edges and N represents the number of nodes. Hence, we can conclude that each iteration of GA takes $O(|E|*log|N|)$ run-

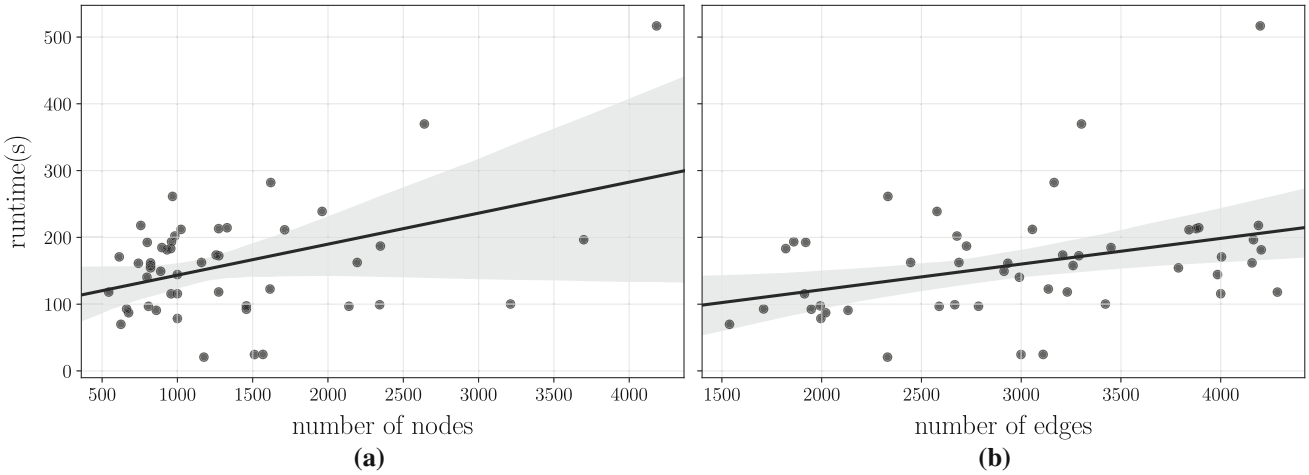


Fig. 15 Runtime analysis of GA. **a** and **b** show the relationship between the runtime and the number of nodes/edges, respectively. Logistic regression model is used to fit the growth curve. In general, the runtime of GA is linear with $|N|$ and $|E|$

time. However, it is hard to predict the number of iterations of GA beforehand. Therefore, we compute the runtime on 49 networks, aiming at giving a general estimation of GA's time complexity. In Fig. 15, we show the relationship between the runtime and the number of nodes or edges, respectively. We also use logistic regression model to fit the growth curve. It can be seen that except for a few extreme cases, the runtime of GA is basically linear with $|N|$ and $|E|$, which ensures GA scaling up quite well.

5 Conclusions

Network dismantling plays an important role the in the evaluation of network robustness. However, finding the optimal attack strategy is an NP-hard problem, existing methods often use node centralities or tailored heuristics to measure the importance of nodes to seek the attack strategies. Experimental results show that these methods cannot achieve a good balance between the quality of the results and the runtime: BI can obtain high-quality result, but at the same time, it has the highest computational complexity of at least cubic in the number of nodes; other methods can be executed much faster than BI, but with inferior attacking quality.

In this study, we proposed a novel network attacking method based on genetic algorithms. We developed three types of initial population, which provide good evolutionary input for networks of various topologies. We proposed a set of genetic operators, which have the tendency to place critical nodes to the front of the attack sequence. These operators can expand the search space of GA, and has the potential to improve the quality of the population. To evaluate the performance of an attack, we designed a novel method to compute the R value. Instead of directly computing the R value in a forward way, we proposed to compute it with the process of building up the network in the reverse order of the attack sequence. This backward structure can significantly reduce the time complexity of computing R, requiring only linear time. In order to select various parameters of GA appropriately, we conducted sensitivity analysis on variants of our method, and evaluated their performance from the aspect of runtime and solution quality, respectively. Three representative parameter settings with different patterns are selected. We tested these three GA instances on a wide range of real-world networks. The results showed that our method significantly outperforms the state-of-the-art methods. It can get the attack sequence of rather high quality, while spending much less runtime than BI, which make a good trade-off between quality of results and runtime.

Taken together, our study provides a different perspective on the network dismantling, using GA to generate attacks to the networks. There are still a few limitations in this study. First, for some networks with high symmetry, such as standard grids, GA has rather limited improvement to population in each iteration. Second, we did not consider the non-uniform cost of deleting nodes Ren et al. (2019). In reality, it may require extremely great effort to remove the critical nodes due to special protection measures. Future work could investigate the network dismantling problem under non-uniform cost consideration. Other directions for future work include the use of extended genetic algorithm variants, such as multi-population genetic algorithms Shi et al. (2020) and cellular genetic algorithms Dahi and Alba (2020); the role of communities Zalik and Zalik (2019), Wandelt et al. (2021) for network dismantling could be exploited further as well. Finally, it might be possible to exploit the idea of sub-

network centrality Cerqueti et al. (2020) for efficient network dismantling.

Funding This study is supported by the National Natural Science Foundation of China (Grants No. 61861136005, No. 61851110763, No. 71731001).

Availability of data and material The data underlying this study are available from the corresponding author upon reasonable request.

Declarations

Conflict of interest Author Wei Lin declares that he has no conflict of interest. Author Sebastian Wandelt declares that he has no conflict of interest. Author Xiaoqian Sun declares that she has no conflict of interest.

Code availability The code underlying this study are available from the corresponding author upon reasonable request.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Ahn CW, Ramakrishna RS (2003) Elitism-based compact genetic algorithms. IEEE Trans Evol Comput 7(4):367–385

Albert R, Albert I, Nakarado GL (2004) Structural vulnerability of the north american power grid. Phys Rev E 69(2):025103

Albert R, Barabási A-L (2002) Statistical mechanics of complex networks. Rev Mod Phys 74(1):47

Alsabti K, Ranka S, Singh V (1997) An efficient k-means clustering algorithm

Ash J, Newth D (2007) Optimizing complex networks for resilience against cascading failure. Physica A 380:673–683

Baker JE et al (1987) Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the second international conference on genetic algorithms 206:14–21

Benavoli A, Corani G, Demšar J, Zaffalon M (2017) Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. J Mach Learn Res 18(1):2653–2688

Bonacich P (1972) Factoring and weighting approaches to status scores and clique identification. J Math Sociol 2(1):113–120

Brandes U (2001) A faster algorithm for betweenness centrality. J Math Sociol 25(2):163–177

Braunstein A, Dall'Asta L, Semerjian G, Zdeborová L (2016) Network dismantling. Proc Natl Acad Sci 113(44):12368–12373

Brooker P (2010) Fear in a handful of dust: aviation and the icelandic volcano. Significance 7(3):112–115

Cerqueti R, Clemente GP, Grassi R (2020) Influence measures in subnetworks using vertex centrality. Soft Comput 24:8569–8582

Clemente GP, Cornaro A (2020) A novel measure of edge and vertex centrality for assessing robustness in complex networks. Soft Comput 24:13687–13704

Colizza V, Barrat A, Barthélemy M, Vespignani A (2006) The role of the airline transportation network in the prediction and predictability of global epidemics. Proc Natl Acad Sci 103(7):2015–2020

Conchon, S., Filliâtre, J.-C.: A persistent union-find data structure. In *Proceedings of the 2007 workshop on Workshop on ML*, pages 37–46, (2007)

Corsi, S., Sabelli, C.: General blackout in italy sunday september 28, 2003, h. 03: 28: 00. In: IEEE Power Engineering Society General Meeting, 2004, pages 1691–1702. IEEE, (2004)

Cuadra L, Salcedo-Sanz S, Del Ser J, Jiménez-Fernández S, Geem ZW (2015) A critical review of robustness in power grids using complex networks concepts. Energies 8(9):9211–9265

Dahi ZA, Alba E (2020) The grid-to-neighbourhood relationship in cellular gas: from design to solving complex problems. Soft Comput 24:3569–3589

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

Duijn PA, Kashirin V, Sloot PM (2014) The relative ineffectiveness of criminal network disruption. Sci Rep 4:4238

Freeman LC (1977) A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41

Friedkin NE (1991) Theoretical foundations for centrality measures. Am J Sociol 96(6):1478–1504

Gao J, Liu X, Li D, Havlin S (2015) Recent progress on the resilience of complex networks. Energies 8(10):12187–12210

Holme P, Kim BJ, Yoon CN, Han SK (2002) Attack vulnerability of complex networks. Phys Rev E 65(5):056109

Katz L (1953) A new status index derived from sociometric analysis. Psychometrika 18(1):39–43

Kitsak M, Ganin AA, Eisenberg DA, Krapivsky PL, Krioukov D, Alderson DL, Linkov I (2018) Stability of a giant connected component in a complex network. Phys Rev E 97(1):012309

Lerman K, Ghosh R (2010) Information contagion: An empirical study of the spread of news on digg and twitter social networks. arXiv preprint arXiv:1003.2664

Merico D, Gfeller D, Bader GD (2009) How to visually interpret biological data using networks. Nat Biotechnol 27(10):921–924

Morone F, Makse HA (2015) Influence maximization in complex networks through optimal percolation. Nature 524(7563):65–68

Morone F, Min B, Bo L, Mari R, Makse HA (2016) Collective influence algorithm to find influencers via optimal percolation in massively large social media. Sci Rep 6:30062

Page L, Brin S, Motwani R, Winograd T (1999) The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab

Peters K, Buzna L, Helbing D (2008) Modelling of cascading effects and efficient response to disaster spreading in complex networks. Int J Crit Infrastruct 4(1–2):46–62

Ren X-L, Gleinig N, Helbing D, Antulov-Fantulin N (2019) Generalized network dismantling. Proc Natl Acad Sci 116(14):6554–6559

Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. In *AAAI*

Sabidussi G (1966) The centrality index of a graph. Psychometrika 31(4):581–603

Schneider CM, Moreira AA, Andrade JS, Havlin S, Herrmann HJ (2011) Mitigation of malicious attacks on networks. Proc Natl Acad Sci 108(10):3838–3841

Shi W, Long Xiaoqiu, Li Y, Deng D, Wei Y (2020) Research on the performance of multi-population genetic algorithms with different complex network structures. Soft Comput 24:13441–13459

Smith J, Fogarty TC (1996) Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of IEEE international conference on evolutionary computation*, pages 318–323. IEEE

Strogatz SH (2001) Exploring complex networks. Nature 410(6825):268–276

Sun X, Wandelt S, Linke F (2015) Temporal evolution analysis of the european air transportation system: air navigation route network and airport network. Transportmetrica B: Transport Dynamics 3(2):153–168

Tarjan RE, Vishkin U (1985) An efficient parallel biconnectivity algorithm. SIAM J Comput 14(4):862–874

Tian L, Bashan A, Shi D-N, Liu Y-Y (2017) Articulation points in complex networks. Nat Commun 8(1):1–9

Wandelt S, Shi X, Sun X (2021) Estimation and improvement of transportation network robustness by exploiting communities. Reliabil Eng Syst Safe 206:107307

Wandelt S, Sun X, Feng D, Zanin M, Havlin S (2018) A comparative analysis of approaches to network-dismantling. Sci Rep 8(1):1–15

Wandelt S, Sun X, Zanin M, Havlin S (2017) QRE: quick Robustness Estimation for large complex networks. Futur Gener Comput Syst 83:02

Whitley D (1994) A genetic algorithm tutorial. Stat Comput 4(2):65–85

Yook S-H, Jeong H, Barabási A-L (2002) Modeling the internets large-scale topology. Proc Natl Acad Sci 99(21):13382–13386

Zalik KR, Zalik B (2019) Node attraction-facilitated evolution algorithm for community detection in networks. Soft Comput 23:6135–6143

Zanin M, Lillo F (2013) Modelling the air transport with complex networks: a short review. Eur Physic J Special Topics 215(1):5–21

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