nature machine intelligence

ARTICLES

https://doi.org/10.1038/s42256-020-0177-2

Finding key players in complex networks through deep reinforcement learning

Changjun Fan[1,2], Li Zeng[1], Yizhou Sun[2] ✉ and Yang-Yu Liu[3,4] ✉

Finding an optimal set of nodes, called key players, whose activation (or removal) would maximally enhance (or degrade) a certain network functionality, is a fundamental class of problems in network science. Potential applications include network immunization, epidemic control, drug design and viral marketing. Due to their general NP-hard nature, these problems typically cannot be solved by exact algorithms with polynomial time complexity. Many approximate and heuristic strategies have been proposed to deal with specific application scenarios. Yet, we still lack a unified framework to efficiently solve this class of problems. Here, we introduce a deep reinforcement learning framework FINDER, which can be trained purely on small synthetic networks generated by toy models and then applied to a wide spectrum of application scenarios. Extensive experiments under various problem settings demonstrate that FINDER significantly outperforms existing methods in terms of solution quality. Moreover, it is several orders of magnitude faster than existing methods for large networks. The presented framework opens up a new direction of using deep learning techniques to understand the organizing principle of complex networks, which enables us to design more robust networks against both attacks and failures.

The network, or graph in discrete mathematics, is a common data structure to describe numerous types of interactive systems[1,2], such as the Internet, social media, transportation networks, power grids, food webs and biomolecular networks. Such systems are greatly affected by a small fraction of important nodes, whose activation/removal would significantly improve/degrade certain network functionality. Such important nodes have been named differently depending on their roles in different application scenarios, for example, influential nodes[3,4], vital nodes[5], key player nodes[6] or critical nodes[7]. Hereafter we will simply call them key players.

Finding an optimal set of key players in complex networks has been a long-standing problem in network science, with many real-world applications. Representative examples include (1) destroying communications in a criminal or terrorist network by arresting critical suspects[8], (2) destroying certain critical proteins and neutralizing the corresponding harmful protein complexes for rational drug design[9], (3) planning resource allocation during an evacuation or reestablishing critical traffic routers in transportation networks[10] after a disaster and (4) handling various diffusion phenomena on networks, including both the optimal spreading problem (that is, maximizing the diffusion for influence spreading or viral marketing[11]) and the optimal immunization problem (that is, minimizing diffusion via epidemic control[12], rumour control[11] and network immunization[13]).

Depending on the specific application scenario, we need to define the corresponding measure to quantify the network functionality appropriately. Without loss of generality, hereafter we consider network connectivity as a key proxy for network functionality. After all, almost all network applications are typically designed to be run in a connected environment[7]. Commonly used network connectivity measures include the number of connected components, pairwise connectivity, the size of the giant connected component (GCC), the length of the shortest paths between two certain nodes and so on. In particular, the size of the GCC is a heavily studied connectivity measure[3,14], because it is relevant to both the optimal attack problem and optimal spreading problem (Supplementary Fig. 14). In fact, the optimal attack problem with the objective of minimizing the GCC size is exactly dual to the optimal spreading problem with linear threshold spreading dynamics[3]. (Note that, in general, the optimal attack and spreading problems are not dual to each other.)

Finding an optimal set of key players in general graphs that optimizes nontrivial and hereditary connectivity measures is typically NP-hard[7] (NP, non-deterministic polynomial time). This prohibits exact and scalable solutions of such problems for large-scale networks. Traditional heuristic or approximate algorithms[3,8,14–18] either require substantial problem-specific search or suffer from deteriorated performances. It is often hard to provide a satisfying balance between effectiveness and efficiency. Moreover, most existing methods are ad hoc for specific application scenarios. Those designed for one particular application often fail on many other applications.

Inspired by the recent advances in deep learning techniques for solving combinatorial optimization problems[19–24], here we introduce FINDER (FInding key players in Networks through DEep Reinforcement learning), a generic and scalable deep reinforcement learning framework to find key players in complex networks (see Fig. 1 for a demonstration of its superior performance over existing methods). In particular, FINDER incorporates inductive graph representation learning[25] to represent graph states and actions, and employs a deep *Q* (action quality score) network that combines reinforcement learning and deep neural networks[26–28] to automatically learn the strategy that optimizes the objective. Extensive experiments on various problem settings demonstrate that FINDER significantly outperforms handcrafted heuristics or approximate methods in terms of both solution quality and time complexity. Given that FINDER is trained purely on synthetic graphs generated by toy network models, the learned superior ability in solving complicated real-world problems suggests a new

[1]College of Systems Engineering, National University of Defense Technology, Changsha, China. [2]Department of Computer Science, University of California, Los Angeles, CA, USA. [3]Channing Division of Network Medicine, Brigham and Women's Hospital and Harvard Medical School, Boston, MA, USA. [4]Center for Cancer Systems Biology, Dana-Farber Cancer Institute, Boston, MA, USA. ✉e-mail: yzsun@cs.ucla.edu; yyl@channing.harvard.edu

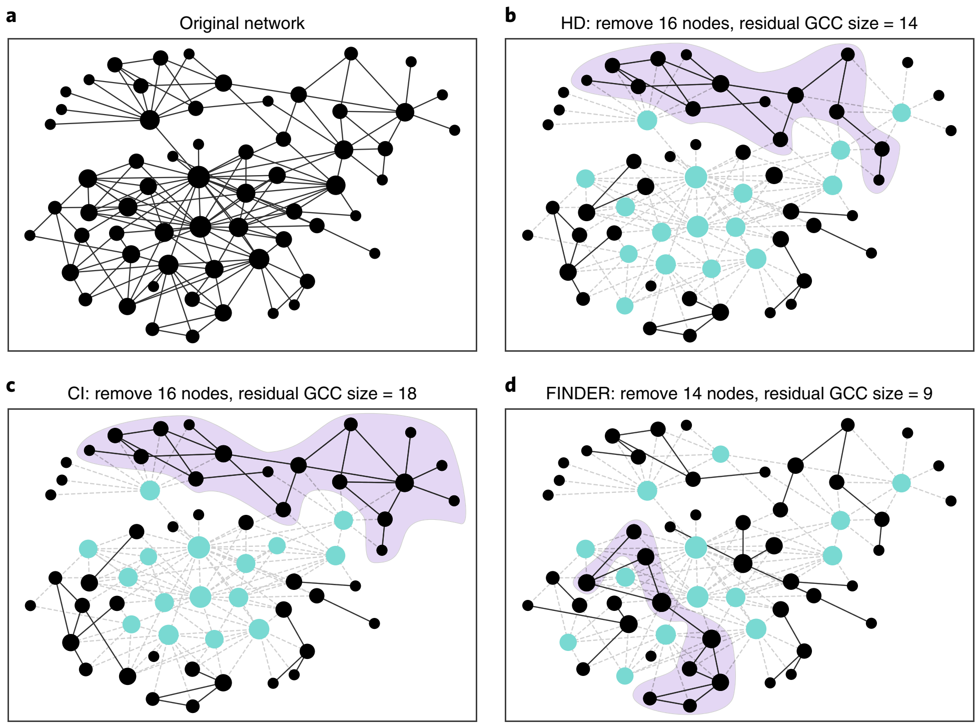
a
Original network
b
HD: remove 16 nodes, residual GCC size = 14
c
CI: remove 16 nodes, residual GCC size = 18
d
FINDER: remove 14 nodes, residual GCC size = 9

Fig. 1 | Finding key players in a network. a, The 9/11 terrorist network[8], which contains 62 nodes and 159 edges. Nodes represent terrorists involved in the 9/11 attack and edges represent their social communications. Node size is proportional to its degree. **b**, Removing 16 nodes (cyan) with the highest degree (HD) causes considerable damage, rendering a remaining GCC (purple) of 14 nodes. **c**, Removing 16 nodes (cyan) with the highest collective influence (CI) results in fragmentation and the remaining GCC (purple) contains 18 nodes. **d**, FINDER removes only 14 nodes (cyan), but leads to a more fragmented network and the remaining GCC (purple) contains only nine nodes. Note that, in the application of maximizing spreading (under linear threshold spreading dynamics with each node's threshold being d_i-1, where d_i is its degree), those key players are not removed but are activated, and the remaining GCC represents inactivated nodes. By minimizing this inactivated GCC in spreading we are effectively maximizing the spreading of information[3].

promising perspective to understand the organizing principles of complex networked systems.

Problem formalization. Formally, given a network $\mathcal{G}=(\mathcal{V},\mathcal{E})$, with a node set $\mathcal{V}$ and an edge set $\mathcal{E}$, and a predefined connectivity measure σ, our learning objective is to design a node removal strategy, that is, a sequence of nodes $(v_1, v_2, ..., v_N)$ to be removed, which minimizes the following accumulated normalized connectivity (ANC)[29]:

$$R(v_1,v_2,\ldots,v_N)=\frac{1}{N}\sum\nolimits_{k=1}^{N}\frac{\sigma(\mathcal{G}\backslash\{v_1,v_2,\ldots,v_k\})}{\sigma(\mathcal{G})} \qquad (1)$$

Here, N is the total number of nodes in $\mathcal{G}$, $v_i\in\mathcal{V}$ denotes the ith node to be removed, $\sigma(\mathcal{G}\backslash\{v_1,v_2,\cdots,v_k\})$ is the connectivity of the residual graph after removing nodes in the set $\mathcal{K}=\{v_1,v_2,\ldots,v_k\}$ sequentially from $\mathcal{G}$, and $\sigma(\mathcal{G})$ is the initial connectivity of $\mathcal{G}$ before any node removal. The value of R can be viewed as an estimation of the area under the ANC curve, which is plotted with the horizontal axis being k/N and the vertical axis being $\sigma(\mathcal{G}\backslash\{v_1,v_2,\ldots,v_k\})/\sigma(\mathcal{G})$. In Fig. 2, we show examples associated with two different connectivity measures, where we apply FINDER to a small real network and plot the ANC curves with three network snapshots highlighted during the node removal procedures.

In certain application scenarios, different nodes are associated with different 'weights', that is, removal costs. We can define a weighted ANC as follows:

$$R_{\text{cost}}(v_1,v_2,\ldots,v_N)=\sum\nolimits_{k=1}^{N}\frac{\sigma(\mathcal{G}\backslash\{v_1,v_2,\ldots,v_k\})}{\sigma(\mathcal{G})}c(v_k) \qquad (2)$$

Here, $c(v_k)$ denotes the normalized removal cost associated with node v_k, and $\sum_{k=1}^{N}c(v_k)=1$. Note that equation (1) is a special case of equation (2), where $c(v_k)=1/N$. The range of both R and R_{cost} lies between 0 and 1 (Supplementary Section IV.B).

In principle, our framework can deal with any well-defined connectivity measure $\sigma:\{\mathcal{G}\}\to\mathbb{R}^+$, which maps a graph into a non-negative real number. To demonstrate the power of our framework, here we consider two most commonly used measures: (1) pairwise connectivity $\sigma_{\text{pair}}(\mathcal{G})=\sum_{C_i\in\mathcal{G}}\frac{\delta_i(\delta_i-1)}{2}$, where C_i is the ith connected component in the current graph $\mathcal{G}$, and δ_i is the size of C_i, which corresponds to the critical node (CN) problem[8]; (2) the size of the GCC, $\sigma_{\text{gcc}}(\mathcal{G})=\max\{\delta_i;C_i\in\mathcal{G}\}$, corresponding to the network dismantling (ND) problem[14], which is also equivalent to the optimal immunization/spreading problem with linear threshold spreading dynamics[3].

Model

Framework. Fundamentally different from traditional methods, FINDER takes a purely data-driven approach without using any domain-specific heuristic. As illustrated in Fig. 3 (top), FINDER is trained offline on small synthetic random graphs generated from classic network models. For each graph, FINDER considers the finding of key players as a Markov decision process: interacting with the environment through a sequence of states, actions and rewards. Here, the environment is the network being analysed, the state is defined as the residual network, the action is to remove or activate the identified key player, and the reward is the decrease of the ANC (equation (1) or equation (2)) after taking the action. During this process, FINDER collects the trial-and-error samples to update its parameters (Supplementary equation (27)) and becomes

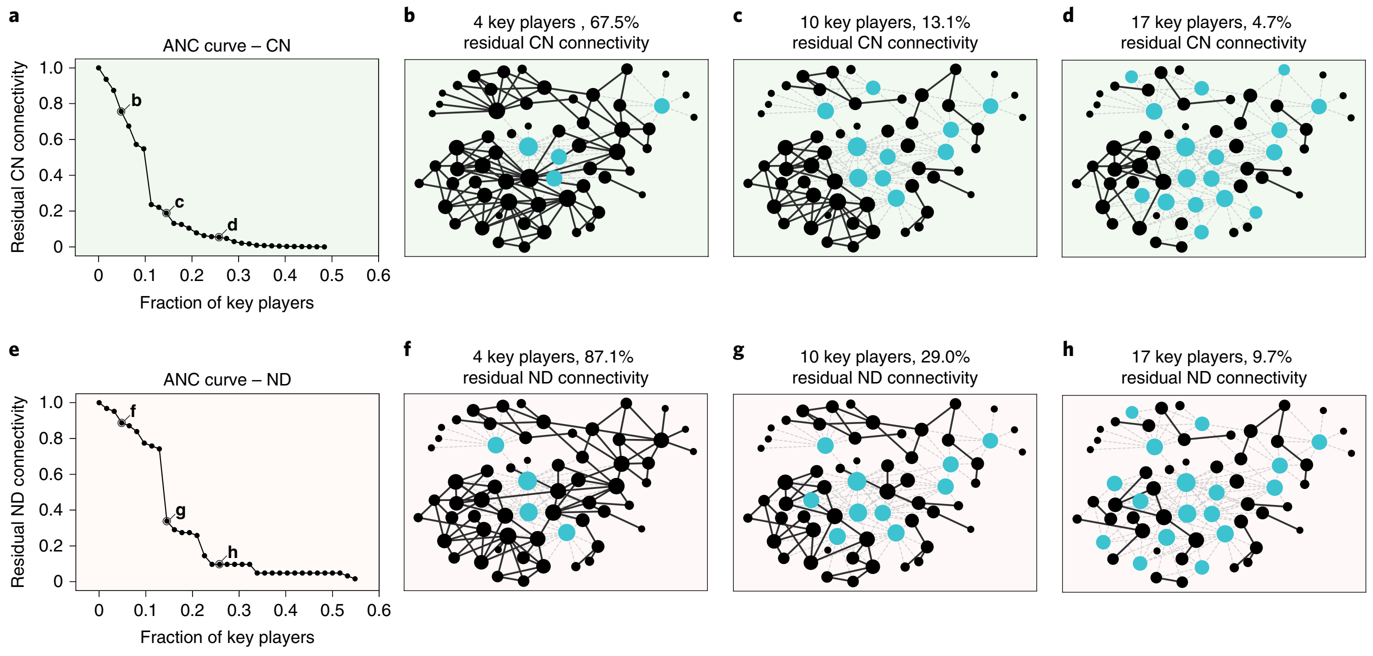

Fig. 2 | The process of finding key players in a network using FINDER. FINDER seeks to design a node removal sequence to minimize the ANC (equation (1) or equation (2)) or, equivalently, minimize the area under the ANC curve, which is generated by sequentially removing the key players identified by FINDER, with the horizontal axis being the fraction of key players and the vertical axis being the network connectivity of the residual graph after removing these key players. **a**,**e**, We consider two connectivity measures for the 9/11 terrorist network (Fig. 1): pairwise connectivity for the critical node (CN) problem (**a**) and GCC size for the network dismantling (ND) problem (**e**). **b**–**d**,**f**–**h**, Residual graphs after removing 4 (**b**,**f**), 10 (**c**,**g**) or 17 (**d**,**h**) key players (cyan) determined by FINDER at the different time points marked in the ANC curves in **a** (**b**–**d**) and **e** (**f**–**h**), respectively.

increasingly intelligent to solve the task (Fig. 3, top). When this offline training phase is over, the well-trained FINDER is able to learn a long-term policy that can select an action to accumulate the maximum rewards from the current state. When applied to a real-world network, FINDER will simply repeat a greedy procedure (Supplementary Section II.D.1) to return the optimal sequence of key players (Fig. 3, bottom).

To ensure success, we still face several challenges. First, how can we represent the states and actions in our setting? Second, how can we leverage these representations to form a score function that tells us the right action for a state? We refer to these two questions as an encoding problem and a decoding problem, respectively.

Encoding. For encoding, traditional methods often use handcrafted features to represent nodes and graphs[30], such as global or local degree distribution, motif counts and so on. However, these features are usually ad hoc and may lead to unsatisfactory performance. Here, we leverage graph representation learning (a.k.a. graph embedding) based on graph neural networks[19,25,31] to characterize the network structural information into a low-dimensional embedding space. In particular, we employ an inductive graph representation learning technique similar to GraphSAGE[25] to iteratively aggregate node embedding vectors, which are initialized as node features (for example, node degree or node removal cost), from the neighbourhood, followed by a nonlinear transformation operator with learnable parameters. After several rounds of recursion, each node obtains an embedding vector that captures both the node's structural location on the graph and the long-range interactions between node features. To capture more complex graph information, we introduce a virtual node that considers all real nodes as neighbours to represent the entire graph[32] and repeat the same embedding propagation process to obtain its representation (see Supplementary Section II.D.1 and Supplementary algorithm 2 for details on encoding).

Decoding. For the decoding, we designed a deep parameterization for the score function, that is, the Q function. The Q function leverages the embeddings of states and actions from the encoder to calculate a score that evaluates the quality of potential actions. Specifically, we apply the outer product operation on embeddings of state and action to model finer state–action dependencies. A multilayer perceptron with rectified linear unit activation is then utilized to map the outer product to a scalar value (see Supplementary Section II.D.1 for decoding details).

Offline training. FINDER was trained over 200,000 randomly generated small synthetic graphs of 30–50 nodes. To perform end-to-end learning of the parameters in the encoder and decoder, we combine the n-step Q-learning loss[19] and the graph reconstruction loss[33] (Supplementary equation (7)) and use Adam gradient descent updates on mini-batch samples, drawn uniformly at random from the pool of stored experiences. The n-step Q-learning loss minimizes the gap between the predicted Q values and target Q values, and the graph reconstruction loss preserves the original network structure in the embedding space. Supplementary algorithm 2 describes the complete training procedure.

Online application. We evaluated FINDER on both synthetic graphs and various real-world networks. During the application phase, we remove a finite fraction of nodes at each adaptive step, instead of the one-by-one removal as in the training phase. We find the performance to be practically unaffected by the removal of up to a 1% fraction (Supplementary Figs. 2 and 3 and Supplementary Tables 6 and 7). This batch nodes selection strategy enables FINDER to scale with $\sim O(|\mathcal{E}| + |\mathcal{V}| + |\mathcal{V}|\log|\mathcal{V}|)$ time complexity (Supplementary Section II.D.3, Supplementary Fig. 4 and Supplementary Table 5), which is very efficient for handling large-scale real-world problems.

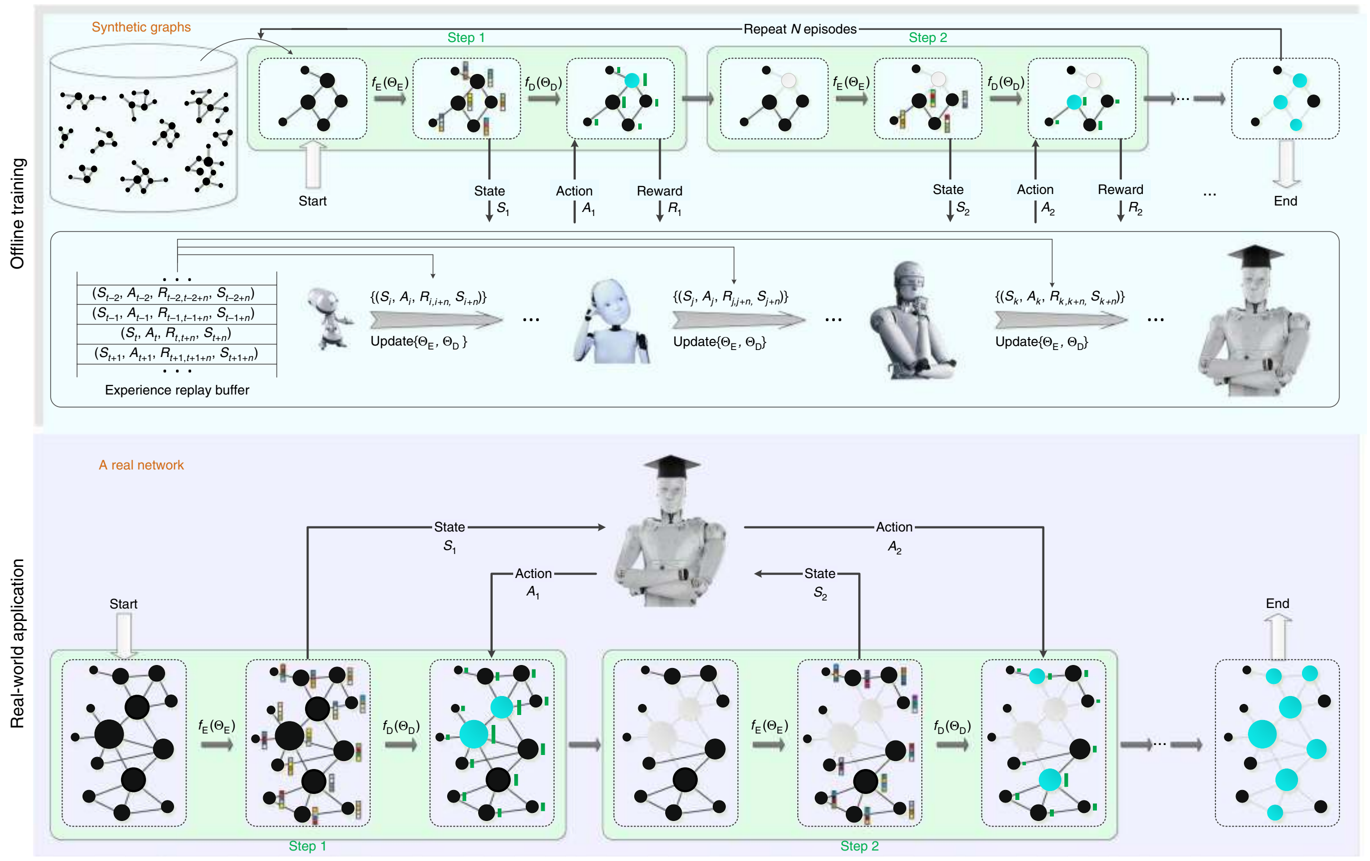

Fig. 3 | Overview of the FINDER framework. The framework consists of two phases: phase 1 (top; offline training, during which the agent FINDER is trained to perform well on synthetic graphs) and phase 2 (bottom; a real-world application, during which the well-trained agent is applied to a real-world network to find the key players). In the offline training, we first generate a batch of synthetic graphs. We then randomly sample one (or mini-batch) of them, and let the FINDER agent 'play the game' on the graph, that is, complete a whole key-player finding process (denoted as an episode), as illustrated in Fig. 2. Specifically, the agent interacts with the graph through a sequence of states, actions and rewards. Here, the state is defined as the residual network, the action is to remove (or activate) the identified key player (node), and the reward is the decrease of ANC after taking the action. To determine the right action for a state, we first encode the current graph and obtain each node's embedding vector (shown as a colour bar), which captures its structure information and long-range interactions between node features (for example, removal cost). We then decode these embedding vectors to scalar Q values (shown as green bars, with heights proportional to the Q values) for all the nodes to predict the long-term gains if taking this action. Based on the calculated Q values, we adopt an ϵ-greedy action strategy; that is, we select the highest-Q node with probability $(1-\epsilon)$ and take a random action otherwise. To balance between exploration and exploitation, ϵ is linearly annealed from 1.0 to 0.05 over 10,000 episodes. When a game (or an episode) is over (for example, the residual graph becomes completely disconnected), we collect the n-step transitions, that is, 4-tuples in the form $(S_i, A_i, R_{(i,i+n)}, S_{(i+n)})$, where $R_{(i,i+n)} = \sum_{k=i}^{i+n} R_k$, from the above sequence, and store them into the experience replay buffer—a queue that maintains the most recent M 4-tuples. In our calculations, we choose $M = 50{,}000$. Meanwhile, the agent is updated (that is, parameters Θ_E and Θ_D for its encoder and decoder are updated) by performing mini-batch gradient descents over the loss (Supplementary equation (7)). As the episodes and updates repeat, the agent becomes increasingly intelligent and powerful in finding key players on complex networks. In the real-world application, once the offline training phase finishes, we can apply the well-trained agent to a real-world network. Here we use the raccoon contact network[39] as an example, and we test on its largest connected component, which contains 14 nodes and 20 edges. Similar to the offline training phase, in the application phase the agent first encodes the current network into low-dimensional embedding vectors, and then leverages these embedding vectors to decode Q values for each node. Unlike the ϵ-greedy action strategy during training, here we exploit the 'batch nodes selection' strategy, which picks a finite fraction (for example, 1%) of highest-Q nodes at each adaptive step, and avoids the one-by-one iterative select-and-recompute of the embedding vectors and Q values. This strategy does not affect the final result, but it renders several orders of magnitude reduction in the time complexity (Supplementary Figs. 2 and 3 and Supplementary Tables 6 and 7). Repeating this process until the network reaches the user-defined terminal state (for example, maximum budget nodes or minimum connectivity threshold), the sequentially removed nodes constitute the optimal set of key players. See Supplementary Section II.D.1 for more details about the framework. Credit: Shutterstock.

Flexibility. We created four FINDER agents to handle two connectivity measures, $\sigma_{\text{pair}}(\cdot)$ and $\sigma_{\text{gcc}}(\cdot)$ (corresponding to CN and ND problems, respectively), under two scenarios: node-unweighted and node-weighted. All the agents share the same architecture (Supplementary Section II.D.1 and Fig. 3), and training procedure (Supplementary algorithm 3), except for the reward function, which is determined by the respective ANC. Extensive experiments demonstrate that our framework is universally effective on these application scenarios, and all FINDER variants that are designed for different problems under different scenarios can converge very well on the validation data (Supplementary Fig. 11). Thanks to its flexible architecture, we anticipate that this framework can be applied to even more complex scenarios as well (see Supplementary Section IV.F, Supplementary Fig. 13 and Supplementary Table 18 for FINDER's adaptation to the minimal percolation threshold problem).

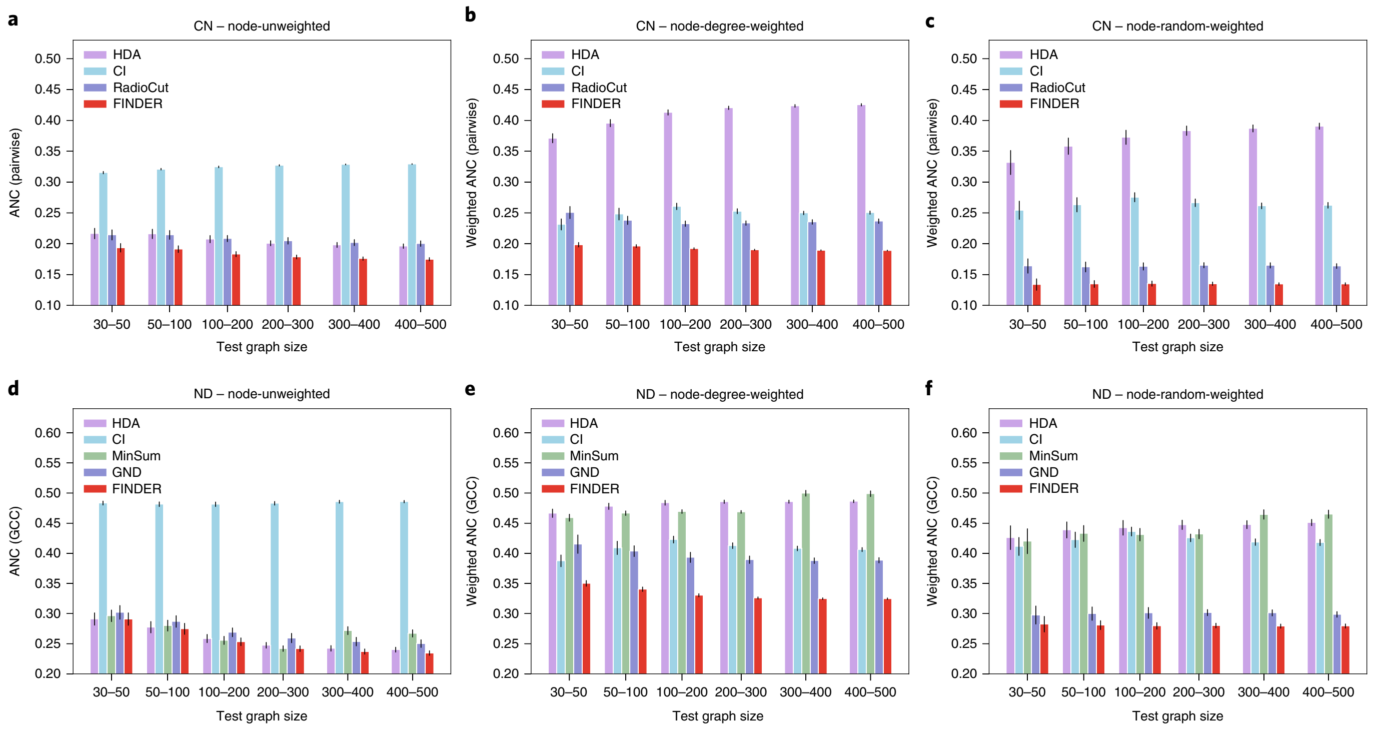

Fig. 4 | Performance of FINDER on synthetic graphs. In all cases, FINDER is trained on BA graphs of 30–50 nodes. We then evaluate the well-trained FINDER on synthetic BA graphs of different scales: 30–50, 50–100, 100–200, 200–300, 300–400 and 400–500 nodes. For each scale, we randomly generated 100 instances, and reported the average results over them. To obtain node-weighted graphs, we assign each node a normalized weight, which is proportional to its degree (degree-weighted) or a random non-negative number (random-weighted). **a**–**c**, Comparison of the results of HDA, CI, RatioCut and FINDER on node-unweighted (**a**), degree-weighted (**b**) and random-weighted (**c**) graphs, respectively, for the CN problem. The results are the averaged ANC determined by the pairwise connectivity over 100 instances. We also compared with other heuristic methods, including HBA, HCA and HPRA (see Supplementary Tables 7–9 for details). **d**–**f**, The results of competing methods and FINDER for the ND problem, on node-unweighted (**d**), degree-weighted (**e**) and random-weighted (**f**) graphs, respectively. The results are the averaged ANC determined by GCC size, over 100 random instances. Comparisons with other baselines, including HBA, HCA, HPRA, BPD and CoreHD, are reported in Supplementary Tables 13–15. It is obvious that FINDER consistently outperforms other methods in different node-weight scenarios for both CN and ND problems. Error bars are the standard deviations over 100 random instances.

Results

Results on synthetic graphs. Figure 4 shows FINDER's performance on synthetic graphs that are significantly larger than those on which it was trained. We first explore the effects of different training graph types. Three classic network models, the Erdős–Rényi (ER) model[34], the Watts–Strogatz (WS)[35] model and the Barabási–Albert (BA) model[36], were used to generate both training and test graphs. As shown in Supplementary Table 20, FINDER performs the best when test and training graphs are generated from the same model. Note that most of the real networks analysed in this work exhibit power-law or fat-tailed degree distributions (Supplementary Fig. 1). Such a high degree heterogeneity is also a key feature of random graphs generated by the BA model. Hence, the agents trained on BA graphs perform consistently better than those trained on ER or WS graphs when tested on various real-world networks (Supplementary Table 21). To empower better generalizations, the agents that were later utilized for different application scenarios were all trained on BA graphs. As shown in Fig. 4, comparing with the state-of-the-art baselines (see Supplementary Section I for details), FINDER trained on small BA graphs consistently achieved better results for both CN and ND problems under different node-weight scenarios in synthetic graphs of much larger sizes.

Results on real-world networks. We then evaluated FINDER on various real-world networks from diverse domains (see Supplementary Section III and Supplementary Table 3 for descriptions). As shown in Fig. 5, FINDER consistently outperforms other methods on most networks in different application scenarios. Especially for node-weighted scenarios, which are more practical and challenging, FINDER excels to a large extent. For the ND node-degree-weighted scenario (Fig. 5k), if we are asked to dismantle Gnutella31 such that the remaining GCC is half of the original size, the current best method (HDA) requires about 40.3% total cost, while our model only needs 14.1%, a reduction of nearly 26.2% cost. If given the same dismantling cost 0.2, the best available method (GND) fragments the network to 80.8% GCC size, while FINDER can achieve up to 35.3%, which is 45.5% better. In addition to the effectiveness advantage, FINDER is also remarkably efficient (Supplementary Tables 10–17), especially on large networks. For example, on the Flickr network, with millions of nodes and tens of millions of edges, FINDER is over 20 times faster than the best performing baseline (GND) for the ND node-degree-weighted scenario (7,734 s versus 174,363 s) and ~890 times faster than the best performing baseline (RatioCut) for the CN node-unweighted scenario (915 s versus 815,411 s). Note that most existing baseline methods do not have GPU implementations, while FINDER can easily be GPU-accelerated. To obtain a fair comparison with baseline methods, we did not deploy GPU acceleration for FINDER in the application phase. (We only utilized GPU to speed up the offline training phase.) Hence, the scalability or efficiency of FINDER presented here is rather conservative.

To further understand the effectiveness of FINDER under node-weighted scenarios, we calculated the cost distributions of the

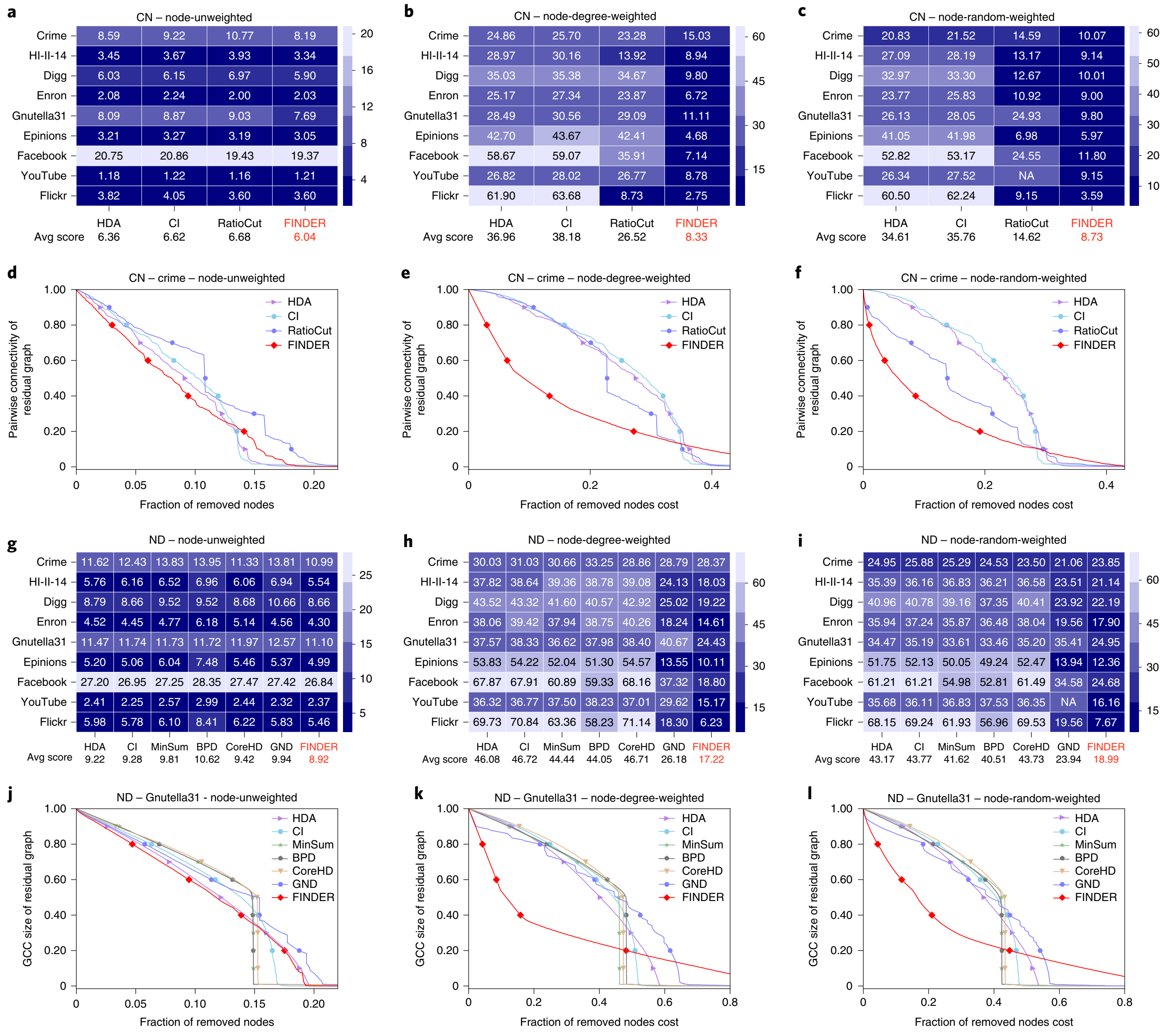

Fig. 5 | Performance of FINDER on real-world networks. We evaluated FINDER on nine real-world networks of five different types—criminal network, biological network, communication network, infrastructure network and social network—representing different applications of key-player finding in these domains. These networks cover a wide range of scales, with node set size ranging from hundreds to millions. The original networks are all node-unweighted. To obtain node-weighted networks, we assigned each node a normalized weight, which is proportional to its degree (degree-weighted) or a random non-negative number (random-weighted). We evaluated FINDER for both CN and ND problems on these networks. **a–c**, ANC results specified by pairwise connectivity for HDA, CI, RatioCut and FINDER in solving CN problems on node-unweighted (**a**), degree-weighted (**b**) and random-weighted (**c**) networks, respectively. **d–f**, ANC curves of these methods on the crime network with different node weights (node-unweighted (**d**), degree-weighted (**e**) and random-weighted (**f**)). For ANC curves for the remaining networks see Supplementary Figs. 5–7. **g–i**, Comparison of the ANC results determined by GCC size, for competing methods with FINDER on the ND problem (node-unweighted (**g**), degree-weighted (**h**) and random-weighted (**i**)). **j–l**, ANC curves of these methods on the Gnutella31 network with node-unweighted (**j**), degree-weighted (**k**) and random-weighted (**l**), respectively. See Supplementary Figs. 8–10 for ANC curves of other real networks on the ND problem. Note that the FINDER utilized here is the same as in Fig. 4 (that is, trained with synthetic BA graphs of 30–50 nodes). All ANC numerical values shown in heatmaps are multiplied by 100 for visualization purposes. We can clearly see that FINDER always produces the best results on these real networks for both CN and ND problems with different node-weight scenarios. Especially for the node-weighted scenario, FINDER shows significant superiority over conventional methods. Running time comparisons on these networks are shown in Supplementary Tables 13 and 17, where we demonstrated the remarkable efficiency advantage of FINDER, especially for large networks.

key players identified by different strategies on the crime network with randomly assigned node weights (removal costs). As shown in Fig. 6, FINDER tends to avoid choosing those 'expensive' key players, which naturally leads to a more cost-effective strategy.

Conclusion

In summary, FINDER achieves superior performances in terms of both effectiveness and efficiency in finding key players in complex networks. It represents a paradigm shift in solving challenging

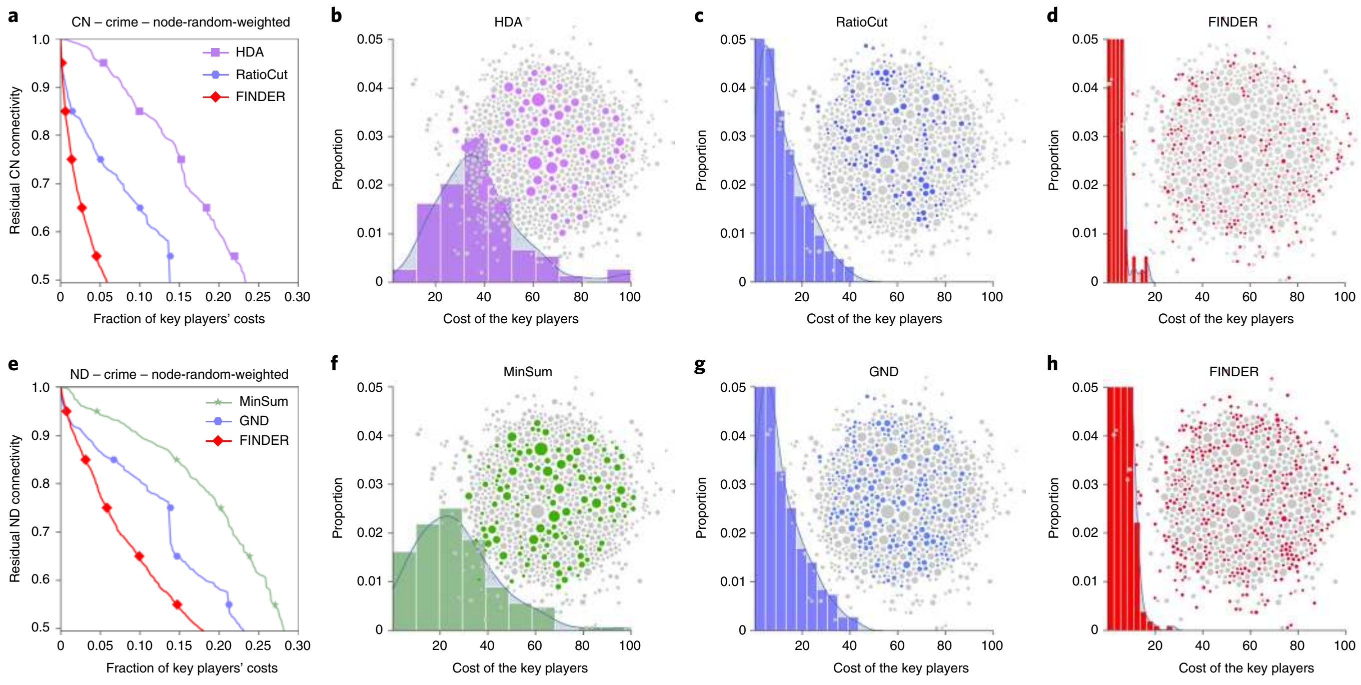

Fig. 6 | Cost distributions of key players identified by FINDER. To further explain the superiority of FINDER over other methods in the node-weighted scenario, we illustrated the cost distributions of the key players. **a**, CN problem on the crime network with random node weights. The ANC is measured by the residual pairwise connectivity with respect to the fraction of removal costs for three methods: HDA, RatioCut and FINDER. **b**–**d**, Distributions of the key-player removal costs and the network with highlighted key players identified by HDA (**b**, purple nodes), RatioCut (**c**, blue nodes) and FINDER (**d**, red nodes), respectively. The target residual pairwise connectivity is set to be 0.5. **e**, ND problem on the crime network with random node weights. The ANC is measured by the residual GCC size with respect to the fraction of removed costs for three methods: MinSum, GND and FINDER. **f**–**h**, Distributions of key-player removal costs and the network with highlighted key players identified by MinSum (**f**, green nodes), GND (**g**, blue nodes) and FINDER (**h**, red nodes), respectively. The target relative residual GCC size is set to be 0.5, that is, 50% of the original network size. For both CN and ND problems, larger nodes denote larger node weights. Conventional methods have higher total costs because they tend to target nodes with higher weights (removal costs), as shown in the histograms. By contrast, FINDER produces a much more cost-effective strategy by avoiding those 'expensive' nodes. The curves in **b**–**d** and **f**–**h** are the kernel density estimates for the key players' cost distributions.

optimization problems on complex networks. Requiring no domain-specific knowledge but just the degree heterogeneity of real networks, FINDER achieves this goal by offline self-training on small synthetic graphs only once for a particular application scenario, and then generalizes surprisingly well across diverse domains of real-world networks with much larger sizes. Thanks to the highly flexible framework of FINDER, for different application scenarios one just needs to replace the rewards with the respective connectivity measures. One can further improve FINDER's performance by tailoring the training data towards the target network with the configuration model (CM)[37] (Supplementary Tables 19 and 22), or by employing the reinsertion technique[38] (Supplementary Fig. 12 and Supplementary Table 9) (see Supplementary Section IV.D for more details about different ways to refine FINDER). Finally, FINDER opens up a new direction of using deep learning techniques to understand the organizing principle of complex networked systems, which enables us to design networks that are more robust against both attacks and failures. The presented results also highlight the importance of classic network models, such as the BA model. Although extremely simple, it captures the key feature, that is, degree heterogeneity, of many real-world networks, which turns out to be extremely useful in solving very challenging optimization problems on complex networks.

Data availability
All the data analysed in this paper, including synthetic graphs and real-world networks, can be accessed through our Code Ocean compute capsule (https://doi.org/10.24433/CO.3005605.v1).

Code availability
All source codes and models (including those that can reproduce all figures and tables analysed in this work) are publicly available through our Code Ocean compute capsule (https://doi.org/10.24433/CO.3005605.v1) or on GitHub (https://github.com/FFrankyy/FINDER).

Received: 6 October 2019; Accepted: 14 April 2020;
Published online: 25 May 2020

References
1. Albert, R. & Barabási, A.-L. Statistical mechanics of complex networks. *Rev. Mod. Phys.* **74**, 47 (2002).
2. Newman, M. E. The structure and function of complex networks. *SIAM Rev.* **45**, 167–256 (2003).
3. Morone, F. & Makse, H. A. Influence maximization in complex networks through optimal percolation. *Nature* **524**, 65–68 (2015).
4. Kempe, D., Kleinberg, J. & Tardos, É. Influential nodes in a diffusion model for social networks. In *International Colloquium on Automata, Languages and Programming* 1127–1138 (Springer, 2005).
5. Corley, H. & David, Y. S. Most vital links and nodes in weighted networks. *Oper. Res. Lett.* **1**, 157–160 (1982).
6. Borgatti, S. P. Identifying sets of key players in a social network. *Comput. Math. Org. Theory* **12**, 21–34 (2006).
7. Lalou, M., Tahraoui, M. A. & Kheddouci, H. The critical node detection problem in networks: a survey. *Comput. Sci. Rev.* **28**, 92–117 (2018).
8. Arulselvan, A., Commander, C. W., Elefteriadou, L. & Pardalos, P. M. Detecting critical nodes in sparse graphs. *Comput. Oper. Res.* **36**, 2193–2200 (2009).
9. Kuntz, I. D. Structure-based strategies for drug design and discovery. *Science* **257**, 1078–1082 (1992).

10. Vitoriano, B., Ortuño, M. T., Tirado, G. & Montero, J. A multi-criteria optimization model for humanitarian aid distribution. *J. Global Optim.* **51**, 189–208 (2011).
11. Kempe, D., Kleinberg, J. & Tardos, É. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 137–146 (ACM, 2003).
12. Pastor-Satorras, R. & Vespignani, A. Epidemic spreading in scale-free networks. *Phys. Rev. Lett.* **86**, 3200 (2001).
13. Cohen, R., Erez, K., Ben-Avraham, D. & Havlin, S. Breakdown of the internet under intentional attack. *Phys. Rev. Lett.* **86**, 3682 (2001).
14. Braunstein, A., Dall'Asta, L., Semerjian, G. & Zdeborová, L. Network dismantling. *Proc. Natl Acad. Sci. USA* **113**, 12368–12373 (2016).
15. Shen, Y., Nguyen, N. P., Xuan, Y. & Thai, M. T. On the discovery of critical links and nodes for assessing network vulnerability. *IEEE/ACM Trans. Netw.* **21**, 963–973 (2013).
16. Mugisha, S. & Zhou, H.-J. Identifying optimal targets of network attack by belief propagation. *Phys. Rev. E* **94**, 012305 (2016).
17. Zdeborová, L., Zhang, P. & Zhou, H.-J. Fast and simple decycling and dismantling of networks. *Sci. Rep.* **6**, 37954 (2016).
18. Ren, X.-L., Gleinig, N., Helbing, D. & Antulov-Fantulin, N. Generalized network dismantling. *Proc. Natl Acad. Sci. USA* **116**, 6554–6559 (2019).
19. Khalil, E., Dai, H., Zhang, Y., Dilkina, B. & Song, L. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems* 6348–6358 (NIPS, 2017).
20. Nazari, M., Oroojlooy, A., Snyder, L. & Takác, M. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems* 9839–9849 (NIPS, 2018).
21. Bello, I., Pham, H., Le, Q. V., Norouzi, M. & Bengio, S. Neural combinatorial optimization with reinforcement learning. Preprint at https://arxiv.org/abs/1611.09940 (2016).
22. Bengio, Y., Lodi, A. & Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. Preprint at https://arxiv.org/abs/1811.06128 (2018).
23. James, J., Yu, W. & Gu, J. Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. In *IEEE Transactions on Intelligent Transportation Systems* 1–12 (IEEE, 2019).
24. Li, Z., Chen, Q. & Koltun, V. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems* 539–548 (NIPS, 2018).
25. Hamilton, W., Ying, Z. & Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems* 1024–1034 (NIPS, 2017).
26. Brown, N. & Sandholm, T. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* **359**, 418–424 (2018).
27. Silver, D. et al. Mastering the game of go without human knowledge. *Nature* **550**, 354–359 (2017).
28. Moravčík, M. et al. Deepstack: expert-level artificial intelligence in heads-up no-limit poker. *Science* **356**, 508–513 (2017).
29. Schneider, C. M., Moreira, A. A., Andrade, J. S., Havlin, S. & Herrmann, H. J. Mitigation of malicious attacks on networks. *Proc. Natl Acad. Sci. USA* **108**, 3838–3841 (2011).
30. Henderson, K. et al. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 1231–1239 (ACM, 2012).
31. Kipf, T. N. & Welling, M. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations* (ICLR, 2017).
32. Lü, L., Zhang, Y.-C., Yeung, C. H. & Zhou, T. Leaders in social networks, the delicious case. *PLoS ONE* **6**, e21202 (2011).
33. Wang, D., Cui, P. & Zhu, W. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 1225–1234 (ACM, 2016).
34. Erdös, P. & Rényi, A. On random graphs. *Publ. Math. Debrecen* **6**, 290–297 (1959).
35. Watts, D. J. & Strogatz, S. H. Collective dynamics of 'small-world' networks. *Nature* **393**, 440–442 (1998).
36. Barabási, A.-L. & Albert, R. Emergence of scaling in random networks. *Science* **286**, 509–512 (1999).
37. Barabási, A.-L. *Network Science* (Cambridge Univ. Press, 2016).
38. Clusella, P., Grassberger, P., Pérez-Reche, F. J. & Politi, A. Immunization and targeted destruction of networks using explosive percolation. *Phys. Rev. Lett.* **117**, 208301 (2016).
39. Rossi, R. A. & Ahmed, N. K. The network data repository with interactive graph analytics and visualization. In *Proceedings of 29th AAAI Conference on Artificial Intelligence* 4292–4293 (ACM, 2015).

Acknowledgements

We are grateful to M. Chen and Z. Liu for the feedback and assistance they provided during the development and preparation of this research. This work is partially supported by NSF III-1705169, NSF CAREER Award 1741634, NSF 1937599, an Okawa Foundation Grant and an Amazon Research Award. C.F. is supported by the CSC Scholarship offered by the China Scholarship Council. Y.-Y.L. is supported by grants from the John Templeton Foundation (award no. 51977) and National Institutes of Health (R01AI141529, R01HD093761, UH3OD023268, U19AI095219 and U01HL089856).

Author contributions

Y.S. and Y.-Y.L. designed and managed the project. Y.S. and C.F. developed the FINDER algorithm. C.F. and L.Z. performed all the calculations. All authors analysed the results. C.F., Y.-Y.L. and Y.S. wrote the manuscript. All authors edited the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information is available for this paper at https://doi.org/10.1038/s42256-020-0177-2.

Correspondence and requests for materials should be addressed to Y.S. or Y.-Y.L.

Reprints and permissions information is available at www.nature.com/reprints.

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

© The Author(s), under exclusive licence to Springer Nature Limited 2020