Published in final edited form as:
*Nat Mach Intell*. 2020 June ; 2(6): 317–324. doi: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], **Yang-Yu Liu**[3,4]

[1]College of Systems Engineering, National University of Defense Technology, Changsha, 410073, China

[2]Department of Computer Science, University of California, Los Angeles, CA, 90024, USA

[3]Channing Division of Network Medicine, Brigham and Women's Hospital and Harvard Medical School, Boston, MA, 02115, USA

[4]Center for Cancer Systems Biology, Dana-Farber Cancer Institute, Boston, MA, 02115, USA

## Abstract

Finding an optimal set of nodes, called key players, whose activation (or removal) would maximally enhance (or degrade) certain network functionality, is a fundamental class of problems in network science[1,2]. Potential applications include network immunization[3], epidemic control[4], drug design[5], and viral marketing[6]. Due to their general NP-hard nature, those problems typically cannot be solved by exact algorithms with polynomial time complexity[7]. Many approximate and heuristic strategies have been proposed to deal with specific application scenarios[1,2,8–12]. 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 influencer finding problems. 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.

Network, or graph in discrete mathematics, is a common data structure to describe numerous types of interactive systems[13,14], e.g., 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, e.g., influential nodes[1,15], vital

**Correspondence** Correspondence and requests for materials should be addressed to Y.S.(yzsun@cs.ucla.edu) and Y.-Y.L. (yyl@channing.harvard.edu).
**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 analyzed the results. C.F., Y.-Y.L. and Y.S. wrote the manuscript. All authors edited the manuscript.

**Competing interests** The authors declare that they have no competing interests.

nodes[16], key player nodes[17], 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 longstanding problem in network science with many real-world applications. Representative ones include: (i) destroy the communications in a criminal or terrorist network by arresting critical suspects[8]; (ii) destroy some critical proteins and neutralize the corresponding harmful protein complexes for rational drug design[5]; (iii) plan the resource allocation during the evacuation or reestablish critical traffic routers after a disaster in transportation networks[18]; (iv) handle various diffusion phenomenon on networks, including both the optimal spreading problem, i.e., maximizing the diffusion for influence spreading or viral marketing[19], and the optimal immunization problem, i.e., minimizing the diffusion via epidemic control[4], rumor control[19], and network immunization[20].

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, etc. In particular, the size of the GCC is a heavily studied connectivity measure[1,2], since it is relevant to both the optimal attack problem and optimal spreading problem (Fig. S14). In fact, the optimal attack problem with the objective of minimizing the GCC size is exactly dual to the optimal spreading problem with the linear threshold spreading dynamics[1]. (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 are typically NP-complete[7]. This prohibits exact and scalable solutions of such problems for large-scale networks. Traditional heuristic or approximate algorithms[1,2,8–12] either require substantial problem-specific search or suffer from deteriorated performances. They are often hard to offer 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 of deep learning techniques in solving combinatorial optimization problems[21–26], here we introduce FINDER (**FI**nding key players in **N**etworks through **DE**ep **R**einforcement learning), a generic and scalable deep reinforcement learning framework to find key players in complex networks (see Fig. 1 for the demonstration of its superior performance over existing methods). In particular, FINDER incorporates inductive graph representation learning[27] to represent graph states and actions, and employs a deep *Q*-network that combines reinforcement learning and deep neural networks[28–30] 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.

Since 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 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 $\sigma$, our learning objective is to design a node removal strategy, i.e., a sequence of nodes $(v_1, v_2, \cdots, v_N)$ to be removed, which minimizes the following *accumulated normalized connectivity* (ANC)[31]:

$$R(v_1, v_2, \cdots, v_N) = \frac{1}{N} \sum_{k=1}^{N} \frac{\sigma(\mathcal{G} \backslash \{v_1, v_2, \cdots, v_k\})}{\sigma(\mathcal{G})} \tag{1}$$

Here, $N$ is the total number of nodes in $\mathcal{G}$, $v_i \in \mathcal{V}$ denotes the $i$-th 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 $\mathcal{K} = \{v_1, v_2, \cdots, v_k\}$ 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 $\sigma(\mathcal{G} \backslash \{v_1, v_2, \cdots, v_k\})/\sigma(\mathcal{G})$. 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", i.e., removal costs. We can define a weighted ANC as follows:

$$R_{\text{cost}}(v_1, v_2, \cdots, v_N) = \sum_{k=1}^{N} \frac{\sigma(\mathcal{G} \backslash \{v_1, v_2, \cdots, v_k\})}{\sigma(\mathcal{G})} c(v_k) \tag{2}$$

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 Eq. 1 is a special case of Eq. 2, where $c(v_k) = 1/N$. The range of both $R$ and $R_{\text{cost}}$ lies between 0 and 1 (SI Sec.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: (i) 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 $i$-th connected component in current graph $\mathcal{G}$, and $\delta_i$ is the size of $C_i$, which corresponds to the critical node (CN) problem[8]; and (ii) size of the GCC $\sigma_{\text{gcc}}(\mathcal{G}) = \max\{\delta_i; C_i \in \mathcal{G}\}$, corresponding to the network dismantling (ND) problem[2], which is also equivalent to the optimal immunization/spreading problem[1].

## 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 offline trained 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 analyzed, 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 (Eq. 1 or Eq. 2) after taking the action. During this process, FINDER collects the trial-and-error samples to update its parameters (Eq. S27), and becomes increasingly intelligent to solve the task (see 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 (SI Sec.II.D.1) to return the optimal sequence of key players (see Fig. 3 (bottom)).

To ensure the 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.

### Enocoding

For the encoding, traditional methods often use handcrafted features to represent nodes and graphs[32], such as global or local degree distribution, motif counts, etc. However, these features are usually ad hoc and may lead to unsatisfactory performance. Here we leveraged graph representation learning (a.k.a. graph embedding) based on graph neural networks[21,27,33] to characterize the network structural information into a low dimensional embedding space. In particular, we employed an inductive graph representation learning technique similar to GraphSAGE[27] to iteratively aggregates node embedding vectors, which are initialized as node features, e.g., node degree or node removal cost, from the neighborhood, and followed by a non-linear transformation operator with learnable parameters. After several rounds of recursion, each node obtains an embedding vector, which 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 introduced a virtual node that considers all real nodes as neighbors to represent the entire graph[34], and repeated the same embedding propagation process to obtain its representation (see SI Sec.II.D.1 and Algorithm S2 for more details on encoding).

### Decoding

For the decoding, we designed a deep parameterization for the score function, i.e., 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 applied the outer product operation on embeddings of state and action to model finer state-action dependencies. Then a multi-layer perceptron with rectified linear unit (ReLU) activation was utilized to map the outer product to a scalar value (see SI Sec.II.D.1 for more decoding details).

## Offline Training

FINDER was trained over 200,000 randomly generated small synthetic graphs of 30–50 nodes. In order to perform the end-to-end learning of the parameters in the encoder and decoder, we combined the $n$-step $Q$-learning loss[21] and the graph reconstruction loss[35] (Eq. S27), and used 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 predicted $Q$ values and target $Q$ values, and the graph reconstruction loss preserves the original network structure in the embedding space. Algorithm S2 describes the complete training procedure.

## Online Application

We evaluated FINDER on both synthetic graphs and various real-world networks. During the application phase, we removed a finite fraction of nodes at each adaptive step, instead of the oneby-one removal in the training phase. We found the performance was practically unaffected by the removal of up to 1% fraction (Fig. S2–S3, Table S6–S7). This *batch nodes selection* strategy enables FINDER scale as ~ $O(E + N + N\log N)$ time complexity (SI Sec.II.D.3, Fig. S4, Table S5), which is very efficient to handle large-scale real-world problems.

## 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 (SI Sec.II.D.1, Fig. 3), and the training procedure (Algorithm S3), 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 (Fig. S11). Thanks to its flexible architecture, we anticipate this framework can be applied to even more complex scenarios as well (see SI Sec.IV.F and Fig. S13, Table S18 for FINDER's adaptation to the minimal percolation threshold problem).

## Results

### Results on Synthetic Graphs

In Fig. 4, we show the FINDER's performances on synthetic graphs that are significantly larger than what they were trained on. We first explored the effects of different training graph types. Three classic network models, Erdős-Rényi (ER) model[36], Watts-Strogatz (WS)[37] model and Barabási-Albert (BA) model[38], were used to generate both training and test graphs. As shown in Table S20, FINDER performs the best when test and training graphs are generated from the same model. Note that most of the real networks analyzed in this work exhibit power-law or fat-tailed degree distributions (Fig. S1). 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 (Table S21). 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-theart baselines (see SI Sec.I for details), FINDER trained on small BA graphs have consistently achieved better results for both CN and ND problems under different node weights 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 SI Sec.III and Table S3 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. Take the ND-node-degree-weighted scenario (Fig. 5k) as an example, 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%, reducing near 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 be up to 35.3%, which is 45.5% better. In addition to the effectiveness advantage, FINDER is also remarkably efficient (Table S10–S17), especially on large networks. For example, on 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 (7,734s vs 174,363s), and about 890 times faster than the best performing baseline (RatioCut) for the CN node-unweighted scenario (915s vs 815,411s). Note that most existing baseline methods do not have GPU implementations, while FINDER can be easily GPU-accelerated. To have a fair comparison with baseline methods, here 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 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 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. And one can further improve FINDER's performance by tailoring the training data towards the target network with the configuration model (CM)[39] (Table S19, Table S22), or by employing the reinsertion technique[40] (Fig. S12, Table S9) (see SI Sec.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 more robust networks against both attacks and failures. The presented results also highlight the importance of classic network models, such as the BA model. Though extremely simple, it captures the key feature, i.e., degree heterogeneity, of many real-world networks, which tends out to be extremely useful in solving very challenging optimization problems on complex networks.

## Data availability

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

## Code availability

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

## Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

## Acknowledgements

We are grateful to Muhao Chen and Zhong Liu for the feedback and assistance they provided during the development and preparation of this research.

## References

1. Morone F & Makse HA Influence maximization in complex networks through optimal percolation. Nature 524, 65 (2015). [PubMed: 26131931]

2. Braunstein A, Dall'Asta L, Semerjian G & Zdeborová L Network dismantling. Proceedings of the National Academy of Sciences 113, 12368–12373 (2016).

3. Chen Y, Paul G, Havlin S, Liljeros F & Stanley HE Finding a better immunization strategy. Physical Review Letters 101, 058701 (2008). [PubMed: 18764435]

4. Pastor-Satorras R & Vespignani A Epidemic spreading in scale-free networks. Physical Review Letters 86, 3200 (2001). [PubMed: 11290142]

5. Kuntz ID Structure-based strategies for drug design and discovery. Science 257, 1078–1082 (1992). [PubMed: 1509259]

6. Richardson M & Domingos P Mining knowledge-sharing sites for viral marketing. In Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 61–70 (ACM, 2002).

7. Lalou M, Tahraoui MA & Kheddouci H The critical node detection problem in networks: a survey. Computer Science Review 28, 92–117 (2018).

8. Arulselvan A, Commander CW, Elefteriadou L & Pardalos PM Detecting critical nodes in sparse graphs. Computers & Operations Research 36, 2193–2200 (2009).
9. Shen Y, Nguyen NP, Xuan Y & Thai MT On the discovery of critical links and nodes for assessing network vulnerability. IEEE/ACM Transactions on Networking 21, 963–973 (2013).
10. Mugisha S & Zhou H-J Identifying optimal targets of network attack by belief propagation. Physical Review E 94, 012305 (2016). [PubMed: 27575146]
11. Zdeborová L, Zhang P & Zhou H-J Fast and simple decycling and dismantling of networks. Scientific Reports 6, 37954 (2016). [PubMed: 27897223]
12. Ren X-L, Gleinig N, Helbing D & Antulov-Fantulin N Generalized network dismantling. Proceedings of the National Academy of Sciences 116, 6554–6559 (2019).
13. Albert R & Barabási A-L Statistical mechanics of complex networks. Reviews of Modern Physics 74, 47 (2002).
14. Newman ME The structure and function of complex networks. SIAM Review 45, 167–256 (2003).
15. 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).
16. Corley H & David YS Most vital links and nodes in weighted networks. Operations Research Letters 1, 157–160 (1982).
17. Borgatti SP Identifying sets of key players in a social network. Computational & Mathematical Organization Theory 12, 21–34 (2006).
18. Vitoriano B, Ortuño MT, Tirado G & Montero J A multi-criteria optimization model for humanitarian aid distribution. Journal of Global Optimization 51, 189–208 (2011).
19. 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).
20. Cohen R, Erez K, Ben-Avraham D & Havlin S Breakdown of the internet under intentional attack. Physical Review Letters 86, 3682 (2001). [PubMed: 11328053]
21. Khalil E, Dai H, Zhang Y, Dilkina B & Song L Learning combinatorial optimization algorithms over graphs. In Advances in Neural Information Processing Systems 30, 6348–6358 (2017).
22. 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 (2018).
23. Bello I, Pham H, Le QV, Norouzi M & Bengio S Neural combinatorial optimization with reinforcement learning. Preprint at https://arxiv.org/abs/1611.09940 (2016).
24. 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).
25. James J, Yu W & Gu J Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. IEEE Transactions on Intelligent Transportation Systems (2019).
26. 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 (2018).
27. Hamilton W, Ying Z & Leskovec J Inductive representation learning on large graphs. In Advances in Neural Information Processing Systems, 1024–1034 (2017).
28. Brown N & Sandholm T Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. Science 359, 418–424 (2018). [PubMed: 29249696]
29. Silver D et al. Mastering the game of go without human knowledge. Nature 550, 354 (2017). [PubMed: 29052630]
30. Moravčík M et al. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. Science 356, 508–513 (2017). [PubMed: 28254783]
31. Schneider CM, Moreira AA, Andrade JS, Havlin S & Herrmann HJ Mitigation of malicious attacks on networks. Proceedings of the National Academy of Sciences 108, 3838–3841 (2011).
32. 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).

33. Kipf TN & Welling M Semi-supervised classification with graph convolutional networks. In Int. Conf. on Learn. Represent.(ICLR) (2017).

34. Lü L, Zhang Y-C, Yeung CH & Zhou T Leaders in social networks, the delicious case. PLOS One 6, e21202 (2011). [PubMed: 21738620]

35. 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).

36. Erdős P & Rényi A On random graphs i. Publ. Math. Debrecen 6, 290–297 (1959).

37. Watts DJ & Strogatz SH Collective dynamics of ‘small-world’networks. Nature 393, 440 (1998). [PubMed: 9623998]

38. Barabási A-L & Albert R Emergence of scaling in random networks. Science 286, 509–512 (1999). [PubMed: 10521342]

39. Barabási A-L Network science (Cambridge university press, 2016).

40. Clusella P, Grassberger P, Pérez-Reche FJ & Politi A Immunization and targeted destruction of networks using explosive percolation. Physical Review Letters 117, 208301 (2016). [PubMed: 27886508]

41. Rossi RA & Ahmed NK The network data repository with interactive graph analytics and visualization. In Twenty-Ninth AAAI Conference on Artificial Intelligence (2015). URL http://networkrepository.com.

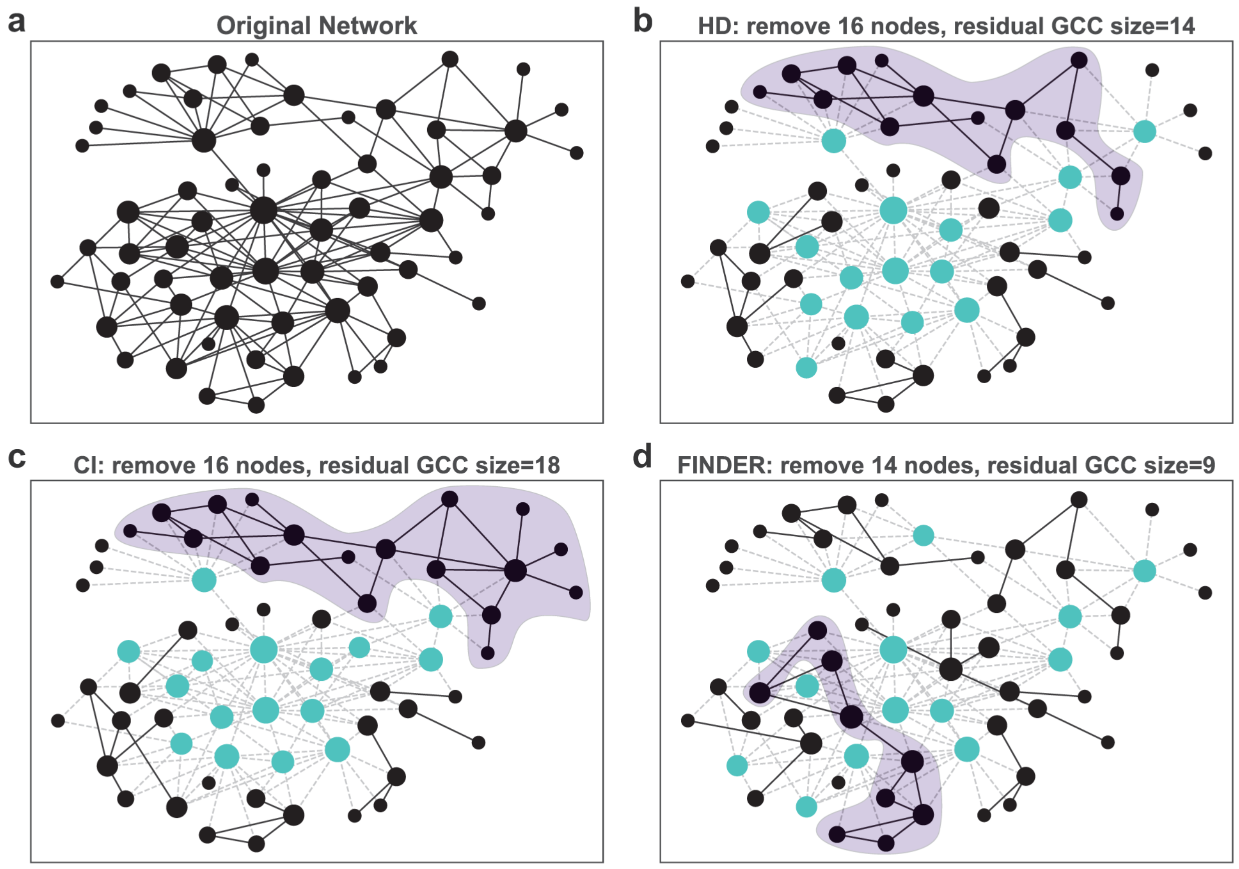
a
Original Network
b
HD: remove 16 nodes, residual GCC size=14
c
CI: remove 16 nodes, residual GCC size=18
d
FINDER: remove 14 nodes, residual GCC size=9

**Figure 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 the considerable damage, rendering a remaining GCC (purple) of 14 nodes. (**c**) Removing 16 nodes (cyan) with the highest collective-influence (CI) results in a 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 9 nodes. Note that in the application of maximizing spreading (under linear threshold spreading dynamics with each node's threshold being $d_i-1$, and $d_i$ 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[1].

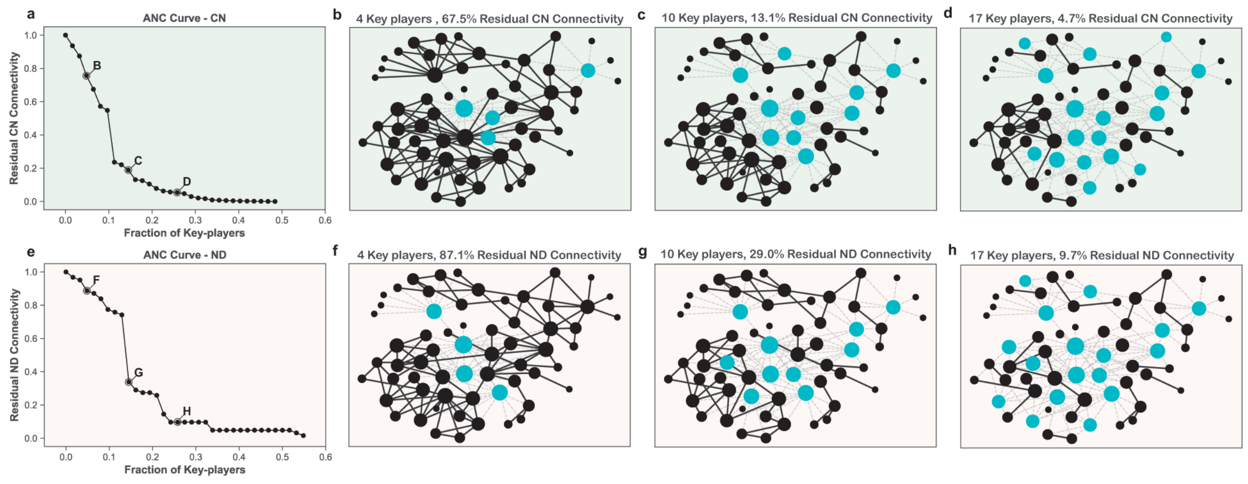

**Figure 2. The process of finding key players in a network using FINDER.**

FINDER seeks to prioritize the key players, or design a node removal sequence, to minimize the ANC (Eq. 1 or Eq. 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. Here we consider two connectivity measures for the 9/11 terrorist network (Fig. 1), i.e., (**a**) pairwise connectivity for the CN problem and, (**e**) GCC size for the ND problem. (**b-d**) or (**f-h**) The residual graphs after removing the key players (cyan) determined by FINDER at different time points marked in the ANC curve in (**a**) or (**e**), respectively.

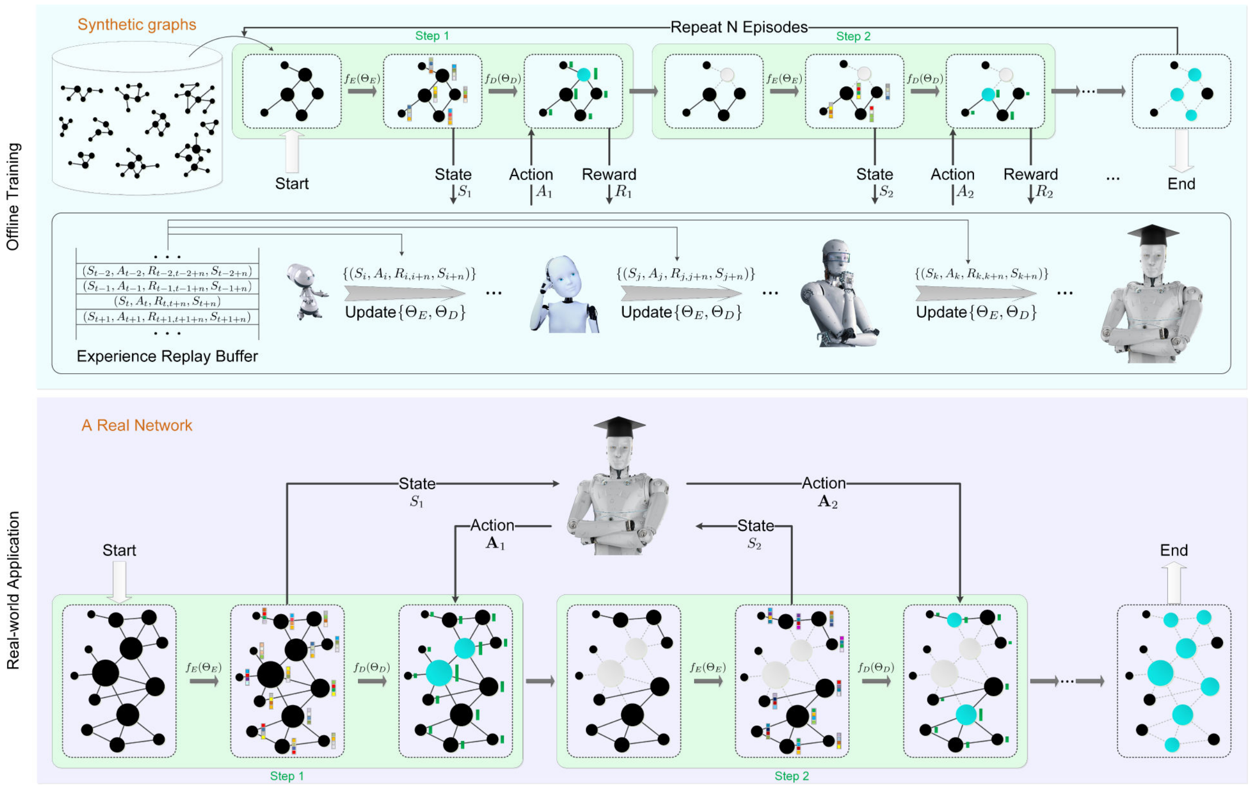
Offline Training
Synthetic graphs
Repeat N Episodes
Step 1
Step 2
$f_E(\Theta_E)$
$f_D(\Theta_D)$
Start
State $S_1$
Action $A_1$
Reward $R_1$
State $S_2$
Action $A_2$
Reward $R_2$
...
End
$(S_{t-2}, A_{t-2}, R_{t-2,t-2+n}, S_{t-2+n})$
$(S_{t-1}, A_{t-1}, R_{t-1,t-1+n}, S_{t-1+n})$
$(S_t, A_t, R_{t,t+n}, S_{t+n})$
$(S_{t+1}, A_{t+1}, R_{t+1,t+1+n}, S_{t+1+n})$
Experience Replay Buffer
$\{(S_i, A_i, R_{i,i+n}, S_{i+n})\}$
Update$\{\Theta_E, \Theta_D\}$
$\{(S_j, A_j, R_{j,j+n}, S_{j+n})\}$
Update$\{\Theta_E, \Theta_D\}$
$\{(S_k, A_k, R_{k,k+n}, S_{k+n})\}$
Update$\{\Theta_E, \Theta_D\}$
Real-world Application
A Real Network
State $S_1$
Action $\mathbf{A}_1$
State $S_2$
Action $\mathbf{A}_2$
Start
End
Step 1
Step 2

**Figure 3. Overview of the FINDER framework.**

The whole framework consists of two phases: Phase-1: Offline training, during which the agent FINDER is trained to perform well on synthetic graphs; Phase-2: Real-world application, during which the well-trained agent is applied to a real-world network to find the key players. (top) Offline Training. We first generate a batch of synthetic graphs. Then we randomly sample one (or mini-batch) of them, and let the FINDER agent 'play the game' on the graph, i.e., complete a whole influencer 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 influencer (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 color bar), which captures its structure information and long-range interactions between node features (e.g., removal cost). Then we decode these embedding vectors to scalar $Q$ values (shown as green bars, with heights proportional to $Q$ values) for all the nodes to predict the long-term gains if taking this action. Based on the calculated $Q$ values, we adopt an $\epsilon$-greedy action strategy, i.e., we select the highest-$Q$ node with probability (1-$\epsilon$) and take a random action otherwise. To balance between exploration and exploitation, $\epsilon$ is linearly annealed from 1.0 to 0.05 over 10,000 episodes. When a game (or an episode) is over (e.g., the residual graph becomes completely disconnected), we collect the $n$-step transitions, i.e., 4-tuples in the form of ($S_i$, $Ai$, $R_{(i,i+n)}$, $S_{(i+n)}$), 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 gets updated (i.e., the parameters $\Theta_E$ and $\Theta_D$ for its encoder and decoder are updated) by performing the mini-batch gradient descents over the loss (Eq. S27). As the episodes and updates repeat, the agent gets increasingly intelligent and powerful in finding key players on complex networks. (**bottom**) 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 mammalia-raccoon network[41] 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 $\epsilon$-greedy action strategy during training, here we exploit the *batch nodes selection* strategy, which picks a finite fraction (e.g., 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 (Fig. S2–S3, Table S6–S7). Repeating this process until the network reaches the user-defined terminal state (e.g., maximum budget nodes or minimum connectivity threshold), the sequentially removed nodes constitute the optimal set of key players. See SI Sec.II.D.1 for more details about the framework. Image credit: copyright, four robot images (from left to right) in the offline training phase, are used under license from Shutterstock.com.

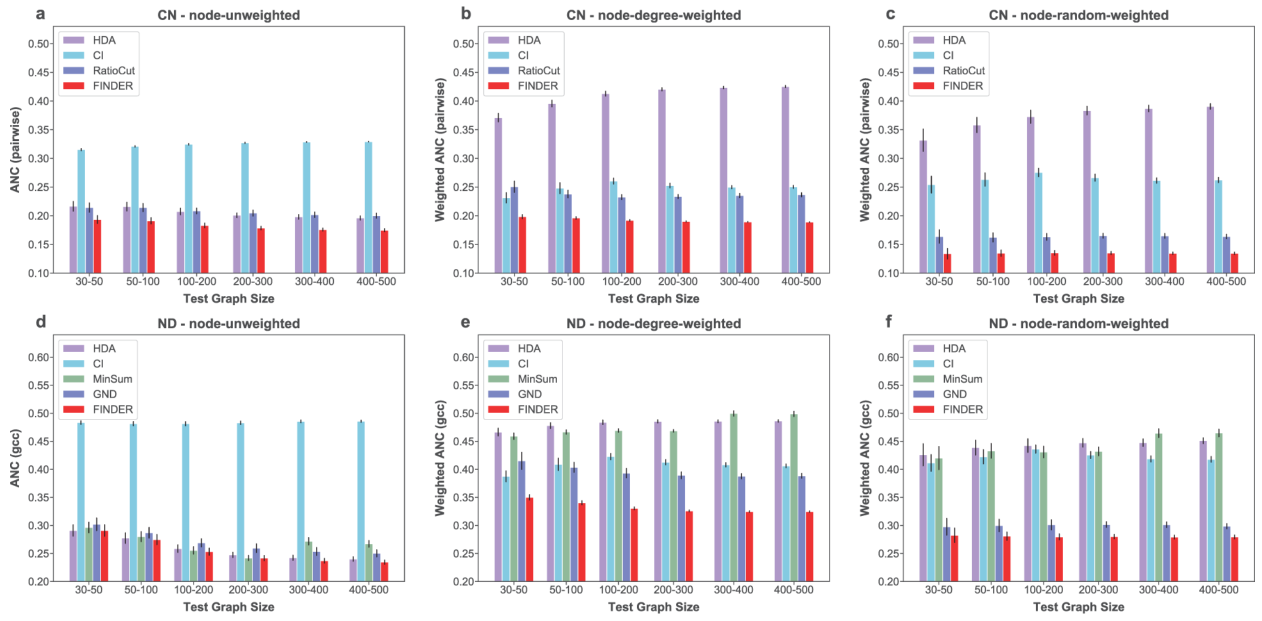

**Figure 4. Performance of FINDER on synthetic graphs.**

In all the cases, FINDER was 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 generate 100 instances, and report 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**) compare the results of HDA, CI, RatioCut and FINDER on node-unweighted, degree-weighted, and random-weighted graphs, respectively, for the CN problem. The results are the averaged ANC determined by the pairwise connectivity, over 100 instances. We also compare with other heuristic methods, including HBA, HCA and HPRA, see Table S7–S9 for details. (**d-f**) illustrate the results of competing methods and FINDER for the ND problem, on node-unweighted, degree-weighted and random-weighted graphs, respectively. The results are the averaged ANC determined by the GCC size, over 100 random instances. The comparisons with other baselines, including HBA, HCA, HPRA, BPD, and CoreHD, see Table S13–S15. It is obvious that FINDER consistently outperforms other methods in different node weight scenarios for both CN and ND problems.

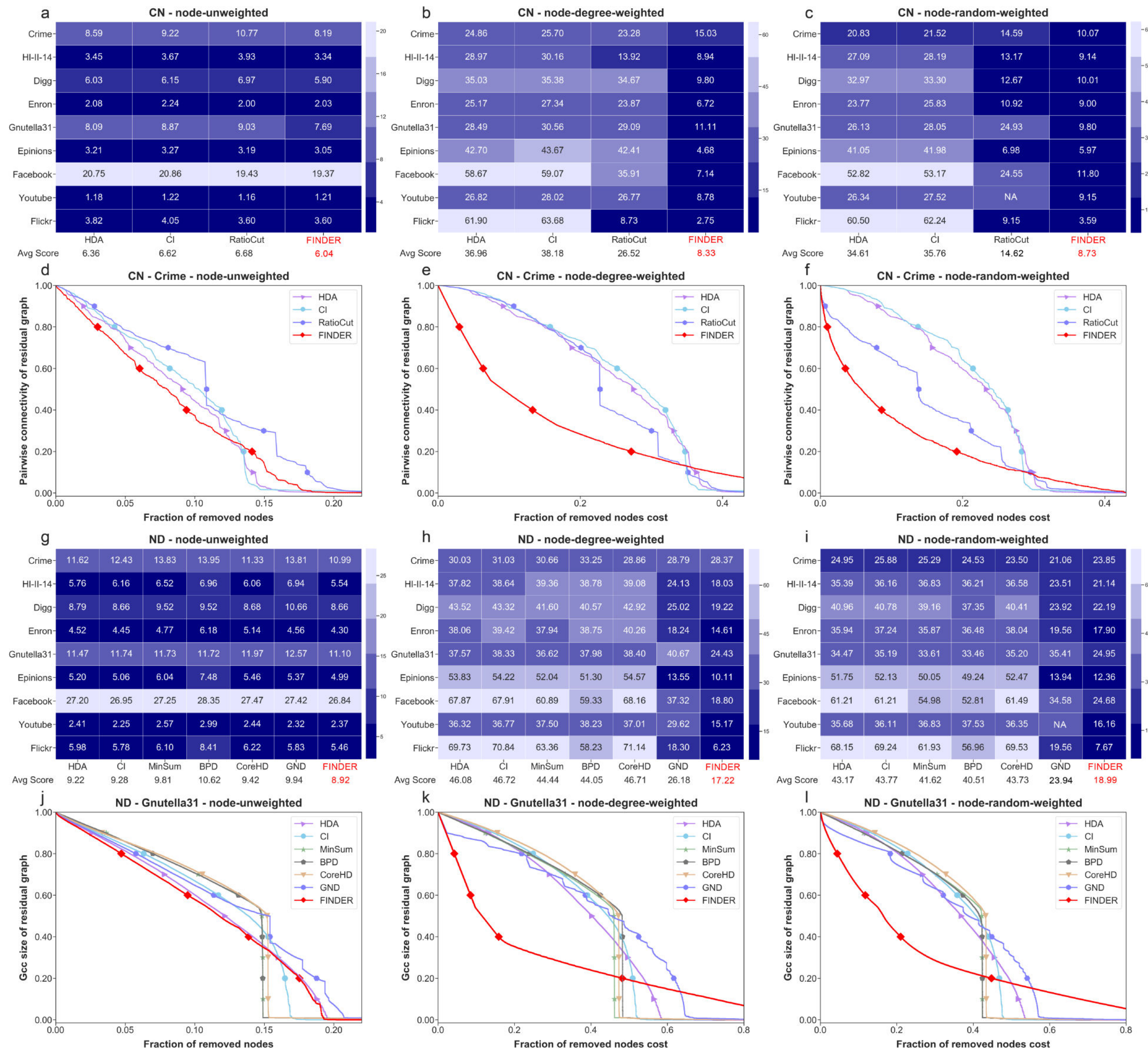

**Figure 5. Performance of FINDER on real-world networks.**

We evaluate 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 influencer 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 assign each node a normalized weight, which is proportional to its degree (degree-weighted) or a random non-negative number (random-weighted).We evaluate FINDER for both CN and ND problems on these networks. (**a-c**) show the ANC results specified by the pairwise connectivity, for HDA, CI, RatioCut and FINDER in solving CN problems on node unweighted, degree-weighted, and random-weighted networks, respectively. (**d-f**) illustrate the ANC curves of these methods on the Crime network with different node weights. ANC curves for the remaining networks see Fig. S5–S7. (**g-i**) compare the ANC results determined by the GCC size, for competing methods with FINDER on the ND problem. (**j-l**) illustrate the ANC curves of these methods on the Gnutella31 network with node-unweighted, degree-weighted and random-weighted, respectively. See Fig. S8–S10 for ANC curves of other real networks on the ND problem. Note that the FINDER utilized here is the same as in Fig. 4, i.e., trained with synthetic BA graphs of 30–50 nodes. All ANC numerical values shown in heatmaps are multiplied by 100 for visualization purpose. 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 Table S13 and S17, where we demonstrate remarkable efficiency advantage of FINDER, especially for large networks.

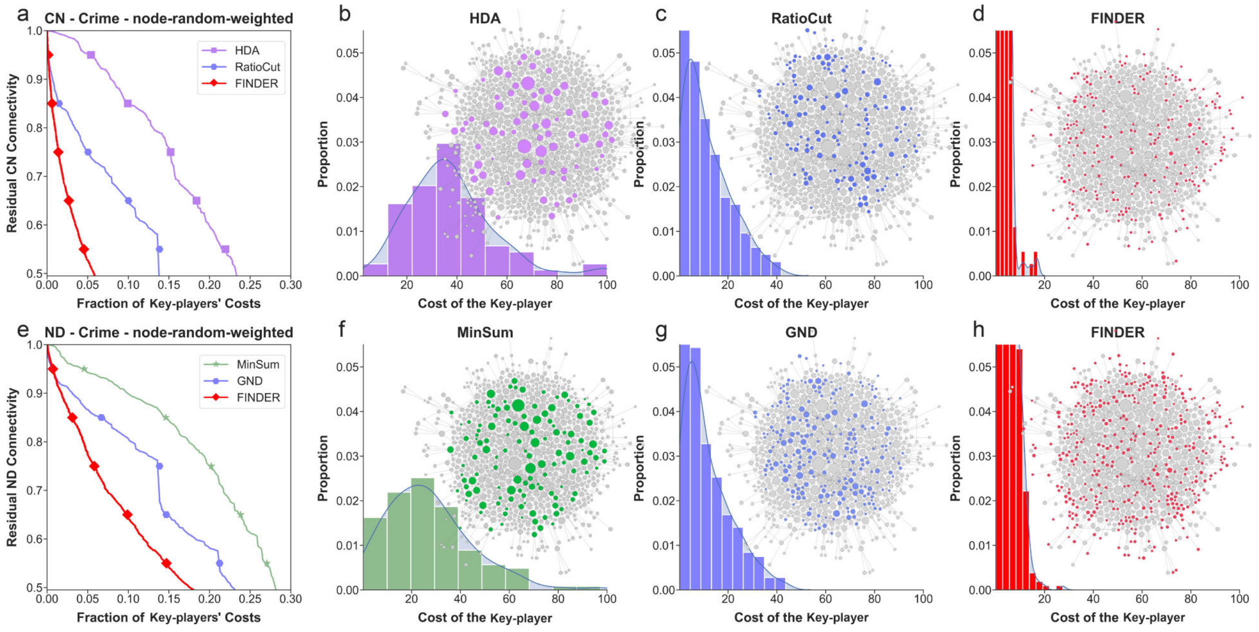

**Figure 6. Cost distributions of key players identified by FINDER.**

To further explain the superiority of FINDER over other methods in the node-weighted scenario, we illustrate 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: HAD, RatioCut, and FINDER. (**b-d**) The distributions of the influencer removal costs and the network with highlighted key players identified by HDA (purple nodes), RatioCut (blue nodes) and FINDER (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**) The distributions of the influencer removal costs and the network with highlighted key players identified by MinSum (green nodes), GND (blue nodes) and FINDER (red nodes), respectively. The target relative residual GCC size is set to be 0.5, i.e., 50% of the original network size. For both CN and ND problems, larger nodes denote larger node weights. Conventional methods take higher total costs since 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.