


# GPIC: A GPU-based parallel independent cascade algorithm in complex networks

Chang Su(苏畅), Xu Na(那旭), Fang Zhou(周方), and Linyuan Lü(吕琳媛)

**Citation:**Chin. Phys. B, 2025, 34 (**3**): 030204. DOI: 10.1088/1674-1056/adb67c
Journal homepage: http://cpb.iphy.ac.cn; http://iopscience.iop.org/cpb

**What follows is a list of articles you may be interested in**

---

## Dynamic analysis of major public health emergency transmission considering the dual-layer coupling of community-resident complex networks

Peng Yang(杨鹏), Ruguo Fan(范如国), Yibo Wang(王奕博), and Yingqing Zhang(张应青)
Chin. Phys. B, 2024, 33 (**7**): 070206. DOI: 10.1088/1674-1056/ad39c7

## Prediction of collapse process and tipping points for mutualistic and competitive networks with *k*-core method

Dongli Duan(段东立), Feifei Bi(毕菲菲), Sifan Li(李思凡), Chengxing Wu(吴成星), Changchun Lv(吕长春), and Zhiqiang Cai(蔡志强)
Chin. Phys. B, 2024, 33 (**5**): 050201. DOI: 10.1088/1674-1056/ad3341

## Intervention against information diffusion in static and temporal coupling networks

Yun Chai(柴允), You-Guo Wang(王友国), Jun Yan(颜俊), and Xian-Li Sun(孙先莉)
Chin. Phys. B, 2023, 32 (**9**): 090202. DOI: 10.1088/1674-1056/acb9f4

## Robustness measurement of scale-free networks based on motif entropy

Yun-Yun Yang(杨云云), Biao Feng(冯彪), Liao Zhang(张辽), Shu-Hong Xue(薛舒红), Xin-Lin Xie(谢新林), and Jian-Rong Wang(王建荣)
Chin. Phys. B, 2022, 31 (**8**): 080201. DOI: 10.1088/1674-1056/ac6942

## Effect of observation time on source identification of diffusion in complex networks

Chaoyi Shi(史朝义), Qi Zhang(张琦), and Tianguang Chu(楚天广)
Chin. Phys. B, 2022, 31 (**7**): 070203. DOI: 10.1088/1674-1056/ac5985

---

**SPECIAL TOPIC — Computational programs in complex systems**

# GPIC: A GPU-based parallel independent cascade algorithm in complex networks

Chang Su(苏畅)[1], Xu Na(那旭)[1], Fang Zhou(周方)[2], and Linyuan Lü(吕琳媛)[3,†]

[1] *Institute of Fundamental and Frontier Sciences, University of Electronic Science and Technology of China, Chengdu* 611731, *China*
[2] *Hangzhou Innovation Institute of Beihang University, Hangzhou* 310000, *China*
[3] *School of Cyber Science and Technology, University of Science and Technology of China, Hefei* 230026, *China*

(Received 27 November 2024; revised manuscript received 10 February 2025; accepted manuscript online 15 February 2025)

Independent cascade (IC) models, by simulating how one node can activate another, are important tools for studying the dynamics of information spreading in complex networks. However, traditional algorithms for the IC model implementation face significant efficiency bottlenecks when dealing with large-scale networks and multi-round simulations. To settle this problem, this study introduces a GPU-based parallel independent cascade (GPIC) algorithm, featuring an optimized representation of the network data structure and parallel task scheduling strategies. Specifically, for this GPIC algorithm, we propose a network data structure tailored for GPU processing, thereby enhancing the computational efficiency and the scalability of the IC model. In addition, we design a parallel framework that utilizes the full potential of GPU's parallel processing capabilities, thereby augmenting the computational efficiency. The results from our simulation experiments demonstrate that GPIC not only preserves accuracy but also significantly boosts efficiency, achieving a speedup factor of 129 when compared to the baseline IC method. Our experiments also reveal that when using GPIC for the independent cascade simulation, 100–200 simulation rounds are sufficient for higher-cost studies, while high precision studies benefit from 500 rounds to ensure reliable results, providing empirical guidance for applying this new algorithm to practical research.

**Keywords:** complex networks, information spreading, independent cascade model, parallel computing, GPU

**PACS:** 02.70.–c, 64.60.aq, 89.20.Ff

**DOI:** 10.1088/1674-1056/adb67c **CSTR:** 32038.14.CPB.adb67c

## 1. Introduction

With the rise of social media,[1,2] the study of information dissemination in online social networks[3] has become a core issue. The independent cascade (IC) model,[4,5] with its strong ability to describe information dissemination in networks, has attracted much attention. It can help identify vital influencers[6–8] and optimize information dissemination strategies.[9–11] However, the computational complexity of the IC model will increase dramatically with the growth of network size, limiting its application in social networks. For one thing, social network experiments often require real-time feedback and dynamic updates, which is beyond the computational ability of traditional algorithms. For another, in information dissemination research, multiple parameters are usually considered, leading to a significant computational burden that can slow down the progress of research work. For example, when Goldenberg used the IC model to simulate the word-of-mouth propagation,[4] he set 5 parameters, each with 7 values, and this required a total of $7^5$ (16807) information dissemination scenarios to be simulated. If the research was expanded to $N$ nodes, with 100 rounds per scenario, then it would require $N \times 16807 \times 100$ simulations to calculate the information dissemination effect of each individual, which would take more than twenty years of computation on a single machine.

To overcome this challenge, researchers have begun to explore parallel computing technologies in recent years to improve the scalability and efficiency of simulation algorithms. The graphics processing unit (GPU),[12] distinguished by its pronounced parallel processing capabilities and cost-effectiveness, has emerged as a formidable instrument for expediting extensive computational tasks. For instance, in 2012, the AlexNet[13] leveraged GPU to enhance the training velocity, and this helped surpass the capabilities of CPU-based model training, achieving significant success in the realm of computer vision. From then on, acceleration algorithms based on GPUs have become a foundational tool across domains such as artificial intelligence, machine learning, and physics.[14–16] However, within the direction of information dissemination, there is yet to be in-depth research[17–19] on effectively mapping the IC model to the GPU architecture.[20–22] To achieve this, it necessitates a meticulous approach to algorithmic design, ensuring that the full spectrum of GPU's parallel processing prowess is harnessed.

In this study, we propose a GPU-based parallel independent cascade (GPIC) algorithm to accelerate the simulation process, which is more efficient and less time-consuming

[†]Corresponding author. E-mail: linyuan.lv@ustc.edu.cn

© 2025 Chinese Physical Society and IOP Publishing Ltd. All rights, including for text and data mining, AI training, and similar technologies, are reserved.

compared with previous IC algorithms. Specifically, the algorithm improves the efficiency and extensibility of the IC model through optimized network data structure and parallel task scheduling strategies.

The main contributions of this paper include: (i) proposing and implementing a parallel algorithm for the IC model tailored for GPU processing; (ii) designing a network data structure and efficient memory access patterns, as well as decomposing and paralleling tasks to enhance the efficiency of the IC model; (iii) validating the accuracy and efficiency of the algorithm through extensive experiments; (iv) providing application examples of the algorithm in actual information dissemination simulations, and demonstrating its potential in practical research problems.

This efficient GPIC algorithm can be used as a fundamental tool to simulate information spreading and has the potential to promote research on information dynamics in complex networks.

The paper is organized as follows: Section 2 involves the method related to the GPIC algorithm. Section 3 introduces the detailed designs of GPIC, including the new data structure, parallel simulation framework, and optimization strategies. Section 4 evaluates the accuracy and efficiency of GPIC. Section 5 applies GPIC in Monte Carlo simulation. In Section 6, we conclude the study by summarizing our findings, discussing their implications, and suggesting future research directions.

## 2. Method

### 2.1. Independent cascade model

The independent cascade (IC) model,[4,5] as one of the information dissemination models, was first proposed by Goldenberg *et al*. As a probabilistic model, it assumes that the success of node $u$'s attempt to activate its adjacent node $v$ is an event with a probability of $p(u,v)$. Moreover, the probability of an inactive node being activated by a newly active neighbor node is independent of the activation attempts of previous neighbors. In addition, the model also assumes that node $u$ has only one opportunity to activate its neighbor node $v$, regardless of success or failure. After this, although $u$ itself remains active, it no longer has the ability to influence others and such nodes are called active nodes without influence. Following is the activating process of the IC model.

1. Define the initial set of active nodes $A$.

2. At time $t$, the newly activated node $u$ influences its adjacent node $v$ with a success probability of $p(u,v)$. If $v$ has multiple neighbors that have just been activated, these nodes will attempt to activate node $v$ in any order.

3. If node $v$ is successfully activated, at time $t+1$, node $v$ becomes active and will influence its adjacent inactive nodes; otherwise, the state of node $v$ does not change at time $t+1$.

4. This process continues until there is no more influential active node in the network and the spreading process ends.

In the later part of this paper, we will take this IC model implemented with Python as the baseline algorithm and compare it with GPIC to examine the performance of the new algorithm.

### 2.2. GPU and CUDA

**GPU** has emerged as the quintessential option for parallel computation, primarily attributed to its distinctive hardware configuration and innovative design principles. Compared with CPU, GPU has more processing cores that can execute tens of thousands of threads simultaneously, providing a hardware foundation for large-scale parallel processing. Its architecture mainly includes:

**Streaming multiprocessor (SM)** is a processing unit in the GPU, which includes a set of processing cores, registers, shared memory and other supporting structures. Each SM is endowed with the autonomy to concurrently manage and execute a cohort of threads, taking charge of the orchestration and operational execution of the threads it oversees.

**Streaming processor (SP)** is the execution unit within SM. An SM contains multiple SPs, which are the most basic processing units for executing computational tasks.

**Compute unified device architecture (CUDA)** is a parallel computing platform and programming model proposed by NVIDIA. Developers can directly use NVIDIA's GPUs for general computing based on the CUDA API. CUDA's parallel computing model adopts a hierarchical management method to effectively organize and schedule threads through the combination of grid, block and thread to achieve highly flexible parallel processing. In terms of memory access, CUDA provides operation support for multiple memory types, including registers, local memory, shared memory and global memory. This design allows developers to perform targeted classification processing based on the performance requirements of the data and optimize the data access efficiency during the calculation process. In terms of computation, it is achieved through kernel functions. Developers can optimize the kernel function to ensure that both SM and SP participate in the calculation as much as possible, thereby improving the overall computing efficiency.

### 2.3. Datasets

This study compares the efficiency of the proposed GPIC algorithm with baseline algorithm using two different types of artificial networks: Barabási–Albert networks (BA)[23] and Erdős–Rényi networks (ER).[24] Table 1 presents the parameters of the BA and ER networks, where $\langle k \rangle$ is the average degree,[2] $k^*$ is the maximum degree, and $c$ is the clustering coefficient.[25] The power-law exponent of BA networks is set to 2. The specific details are as follows.

**Table 1.** Details of the datasets.

| Dataset | Nodes | Edges | $\langle k \rangle$ ($k^*$) | $c$ |
|---|---|---|---|---|
| BA-1000 | 1000 | 3990 | 7.98 (957) | 0.8607 |
| BA-3000 | 3000 | 11990 | 7.99 (2805) | 0.8550 |
| BA-5000 | 5000 | 19990 | 7.99 (4920) | 0.9838 |
| BA-7000 | 7000 | 27990 | 7.99 (6921) | 0.9883 |
| BA-10000 | 10000 | 39990 | 7.99 (9844) | 0.9906 |
| ER-1000 | 1000 | 3990 | 7.98 (17) | 0.0100 |
| ER-3000 | 3000 | 11990 | 7.99 (21) | 0.0019 |
| ER-5000 | 5000 | 19990 | 7.99 (20) | 0.0018 |
| ER-7000 | 7000 | 27990 | 7.99 (19) | 0.0012 |
| ER-10000 | 10000 | 39990 | 7.99 (21) | 0.0007 |

We also use the Watts–Strogatz (WS) network[26] to examine our algorithm. The WS network has a higher clustering coefficient, and its average degree and maximum degree are similar to that of the ER network, so is the acceleration effect.

## 3. Design of GPIC

This section provides a detailed introduction to the GPIC algorithm. As each active node activates its adjacent nodes independently in the independent cascade model, this activating process can be computed in parallel. And for each simulation, we can also leverage the parallel capabilities of the GPU to accelerate computation.

The algorithm is designed to implement the IC model for performing various network tasks.

**Task 1** In a network with $N$ nodes, select a set of seed nodes and simulate the propagation effect for $R$ rounds. This type of task is commonly used in research such as influence simulation[27] and influence maximization.[21,28]

**Task 2** In a network with $N$ nodes, select a set of seed nodes, immunize some of them, and then compare the propagation effects when selecting different nodes to immunize; repeat the simulation for $R$ rounds. This type of task is commonly used in research such as information dissemination intervention simulation,[29,30] network disintegration[31,32] and network attack.[33]

**Task 3** In a network with $N$ nodes, select each node as a seed node and assess its individual propagation effect; repeat the simulation for $R$ rounds. This type of task is commonly used in research such as assessing the influence of individual nodes[34,35] and mining important nodes.

Note that the activation probability $P$ in the propagation model can be a fixed value for homogeneous propagation or vary with edge weights for heterogeneous propagation.

The algorithm mainly consists of three parts: data structure, parallel simulation, and optimization strategies.

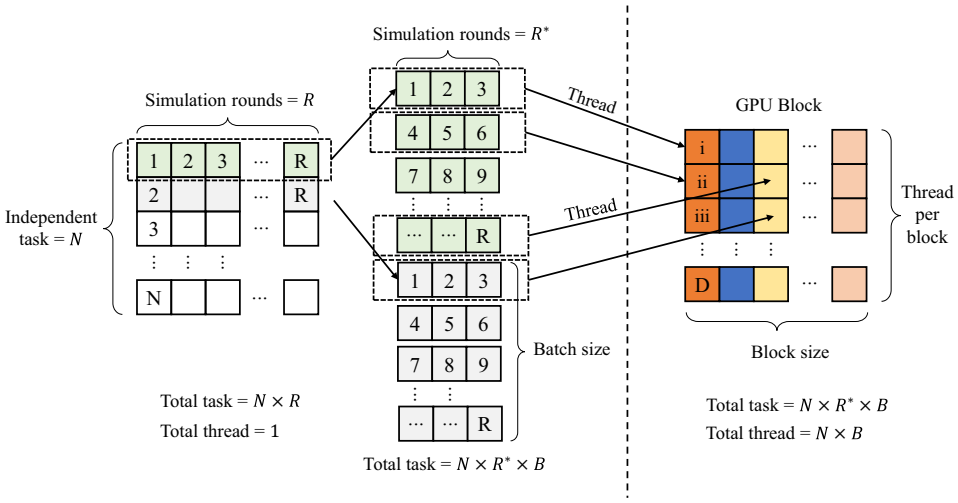



**Fig. 1.** Task decomposition and parallelization. The algorithm divides the $N \times R$ independent tasks into batches and sizes suitable for GPU parallel scheduling. Through block parallelism and thread parallelism, the serial tasks of a single CPU are mapped into parallel computing tasks of $N \times B$ independent threads.

### 3.1. Data structure

Data preprocessing is a fundamental step that involves transforming raw data into a format suitable for the algorithm. The raw network data is generally constructed as edge list data which are composed of node pairs or weighted node pairs. To adapt to the data structure in the GPU and facilitate indexing, we aim to represent network data as an adjacency list that includes the list length: [$\langle$list length, node$_1$, node$_2$, ...$\rangle$]. Compared with the adjacency matrix, which occupies $N \times N$ of graphics card memory space, the adjacency list only occupies $N \times \max(\text{degree})$ of graphics card space. Therefore, for random networks where the maximum degree is usually low, the space usage is only 0.2%–1.7% of the adjacency matrix, saving a significant amount of graphics card memory. However, for scale-free networks with high-degree hub nodes, the space usage is only 2%–7% less than the adjacency matrix.

In response to this, we propose another data structure based on the edge list. We store the network data in the form of an edge list, which will occupy $\mathcal{O}(3E)$ space ([$\langle$node$_1$, node$_2$, weight$\rangle$]), and then construct an index data structure to facilitate the indexing of network data: [$\langle$node id, index start num-

ber, edge count⟩], which will occupy $\mathcal{O}(3N)$ space. For example, for the data used in this study, $E \approx 4N$, it will occupy a total space of $15N$, which is only 0.1%–1.1% of the space occupied by the adjacency matrix. For social networks, when the network density is relatively sparse, the second method is often better than the first one. However, as the number of edges increases, the first method may be better than the second one. For example, in a complete graph, the first method only occupies $N^2 - N$ space, while the second method occupies $\frac{3}{2}N^2 + \frac{3}{2}N$ space.

The actual memory usage is affected by task requirements, network structure (including network size $N$ and maximum degree) and the number of repeated rounds $R$. Among these factors, as task requirements depend on the situation and $R$ is generally fixed at around 50–1000. Thus, it is important to optimize memory usage for network data. The actual measured results under the current configurations are as follows: a network with 10000 nodes occupies 14 MB of memory for Task 1, 4 MB for Task 2, and 178 MB for Task 3 (see Table 2). It can be seen that most network calculations can be performed on consumer-grade graphics cards.

**Table 2.** Memory usage of different tasks.

| Dataset | Task | Repeat rounds | Memory usage (MB) |
|---|---|---|---|
| ER-10000 | Task 1 | 1000 | 14 |
| | Task 2 | 1000 | 4 |
| | Task 3 | 1000 | 178 |
| ER-5000 | Task 1 | 1000 | 8 |
| | Task 2 | 1000 | 2 |
| | Task 3 | 1000 | 66 |

### 3.2. Parallel simulation framework

The implementation of the IC model using a GPU-based parallel algorithm largely depends on the design and implementation of the kernel function. Due to constraints related to graphics card memory space and management mechanisms, the kernel function cannot freely request and release memory space as on the CPU platform. Instead, it must achieve the algorithm's effects within the limited graphics card memory, ideally within the original array space without requiring additional memory allocation. To address this, we propose a method that will additionally apply for a temporary space of size $P \times N$ ($P$ denotes the parallel number) to record the activation status of the current round. The data type of temporary space is represented by an unsigned short integer (i.e., uint8), which could save more memory space compared to other data types such as signed integers and uint32.

For the parallelization of Task 1, the main requirement is to perform multiple rounds of Monte-Carlo simulation with a given seed set. This involves multi-threaded batch parallelism, which requires a thorough consideration of both the block size and thread size. The batch size should not be too small, which would lead to a lot of time being consumed in the creation and destruction of threads. At the same time, the batch size should also not be too large, which would result in a low level of parallelization, failing to achieve the best acceleration effect.

For the parallelization of Task 2, an additional requirement of node immunization is introduced in addition to the operations in Task 1. This new requirement does not require a fundamental change in operations, except for an increase in computational workload: from $R$ rounds of simulation in Task 1 to $R \times G$ (the number of immunized node groups). However, this slight change results in a significant increase in comparisons between experimental groups, leading to an exponential rise in computational workload. This also puts higher demands on computational efficiency.

For the parallelization of Task 3, the amount of computation is $N \times R$. Given that $N$ (representing the network size) could be extremely large, the amount of computation of this task could far exceed those of Task 1 and Task 2. The algorithm carries out different designs on thread task assignment.

### 3.3. Optimization strategies

The algorithm needs to fully utilize the two-level parallelism mechanism inside the GPU, block parallelism and thread parallelism. Given that optimizing thread parallelism efficiency requires optimizing the kernel function to reasonably divide tasks, which requires in-depth analysis of task content, we choose to optimize block parallelism, which only requires choosing a reasonable block size to avoid wasting computing resources.

**Block size** There are two key points for block size optimization.

The specific hardware component that executes the block is the streaming multiprocessor (SM). It is possible to place all threads in the block on the same SM. This can avoid cross-SM calculations, effectively improving memory access speed and computational efficiency.

Threads in the GPU are packaged and executed according to Warp. Generally, the size of a Warp is 32, and threads that are not enough for 32 will also be packaged into one Warp. To avoid resource waste, it is recommended to set the number of threads in multiples of 32.

Optimizing block size in this way for GPU computation may offer minimal efficiency gains for tasks like Task 1 and Task 2, which have low computational requirements. However, for high-computational tasks like Task 3, the optimization can lead to significant improvements in efficiency.

**Batch size** In multi-round repeated simulations, we can divide the $R$ rounds of simulation into $B$ batches, i.e., $R = R^* \times B$. Although the total computational workload remains the same, dividing the task into $B$ batches allows for greater utilization of the GPU's available resources. This approach

increases the number of active SMs and SPs, thereby enhancing computational efficiency. However, it should be noted that increasing the number of batches will require more memory for temporary results, which increases the GPU's memory usage. Therefore, the number of batches should not exceed the GPU's maximum memory capacity, as this would reduce computational efficiency.

### 3.4. Flow diagram

The following pseudocode presents the implementation logic of the parallel algorithm.

**Algorithm 1:** parallel independent cascade algorithm

**Input:** Network structure: *adj(G)*, Activation probability: *wadj(G)*, Origin *seed set* , Repeat times: $R$, Result space: *result* and Temporary space: *temp*

**Output:** activated nodes set size

```
Set current thread = CUDA Block Dim × Block id + CUDA thread id;
Use current thread location the result and temp;
for r = 0; r ≤ R; do
    activated nodes set = set{seed}, this set in Temporary space;
    new activated nodes set = set{seed}, this set in Temporary space;
    while new activated nodes set is not Empty do
        current round activated nodes set = empty set;
        for activated node in new activated nodes set do
            if activated node's neighbour nodes are not Activated then
                try to activate neighbour;
                if activate success then
                    activated nodes set add neighbour node;
                    current round activated nodes set add neighbour node;
        new activated nodes set = current round activated nodes set;
return write the mean of R times activated nodes sets size to result;
```

### 3.5. Usability and extensibility

GPIC offers tool functions to preprocess input data, ensuring it's ready for algorithms. These tools can handle various network structures represented by Python libraries like *networkx* and *igraph*, simplifying the process for researchers. Researchers only need to provide network structure and propagation weights as input to simulate the independent cascade spreading process. This design allows the GPIC algorithm to seamlessly integrate with the established workflow of the traditional IC algorithm, thereby saving time during implementation and operation.

In terms of extensibility, the logic control part of the GPIC algorithm is written in Python. It takes advantage of the characteristics of dynamically interpreted languages and can be run without compilation, making it easy to embed core simulation methods. The core logic code of the extended algorithm can follow the code structure of the Python version. By simply adding the GPU thread number and the code to save the result data, a smooth transition from CPU to GPU can be achieved. However, for spreading models with complicated logic, further design of interaction strategies between threads is required to achieve expansion, which has certain limitations.

## 4. Evaluation of GPIC

To evaluate the performance of the GPIC algorithm, we conduct extensive experiments to compare it with the CPU-based IC algorithm. This section will first introduce the environmental setup, then the metrics for measuring performance, and finally the accuracy and efficiency results.

### 4.1. Experimental setup

#### 4.1.1. Experimental tasks

The parameters that mainly affect the computational complexity of the IC model include network size $N$, spreading probability $P$, and the number of simulation rounds $R$. Several cases are considered to explore how the parameters affect the computational complexity of the IC model. We design three specific testing tasks to measure the performance of the proposed GPIC algorithm.

**Task 1** Identify the top 100 nodes by degree as candidate seed nodes. Add nodes to the seed node set one by one in order of degree, increasing in size from 1 to 100. Implement each simulation for $R$ rounds and analyze the spreading effect of different seed sizes.

**Task 2** Select the top 100 nodes by degree as seed nodes and identify the next largest degree of 1%–20% nodes as can-

didate immune node set. Add these candidate nodes to the immune node set individually in order of degree, increasing in size from 1% to 20%. Implement each simulation for $R$ rounds and study the change of spreading size of different immune sizes.

**Task 3** Select a different node as a seed node. Make each seed node undergo $R$ rounds of simulation and study the spreading size of a single node. This is to explore the relationship between the spreading effect of nodes and network structure and its attributes.

#### 4.1.2. Experimental environment

For the hardware environment, the baseline IC algorithm is conducted on the CPU platform, which uses an Intel Core i9-10900 K@3.70 GHz. The GPIC algorithm is conducted on the GPU platform, which is the main computing environments commonly used by researchers, including laptop, desktops, and servers:

Laptop configuration: Nvidia GeForce MX150 graphics card, 384 CUDA cores, 2 GB GDDR5 memory.

Desktop configuration: Nvidia GTX 1660SP graphics card, 1408 CUDA cores, 6 GB GDDR6 memory.

Server configuration: Nvidia RTX 3090 graphics card, 10496 CUDA cores, 24 GB GDDR6X memory.

For the operating environment, the CPU platform for the baseline algorithm uses Python 3.10.4. The basic operating environment of the GPU platform for GPIC is the same as that of the CPU platform, with the addition of support libraries for GPU programming, including CUDA 12.5 and numba-0.59.1.

### 4.2. Evaluation metrics

We employ the following three metrics to evaluate the accuracy and efficiency of GPIC.

**Mean and variance** This is to test GPIC's accuracy. We conduct a consistency test on the results, analyzing the mean and variance of the spreading sizes produced by both the GPIC and baseline algorithm. By assessing the statistical similarity between these results, we can determine the reliability and accuracy of the GPIC algorithm in simulating spreading processes.

**Execution time** This is to test GPIC's efficiency. It refers to the time required for the algorithm to complete $R$ rounds of IC spreading process. The time for data preparation and preprocessing is not included.

**Speedup effect** This is to test GPIC's efficiency. Take the execution time of the baseline algorithm as the baseline time $t(\mathrm{BL})$, and the execution time of the GPIC algorithm as $t(F)$, and the speedup effect is speedup $= t(\mathrm{BL})/t(F)$.

### 4.3. Experimental results

#### 4.3.1. Accuracy of GPIC

To analyze the accuracy of the proposed GPIC algorithm, we apply the GPIC and baseline algorithm to BA and ER networks with different sizes. Three tasks mentioned above are tested, and the results are displayed in Figs. 2–4. In all the tasks, we find that the mean and variance of the spreading size obtained by the GPIC algorithm are close to those of the baseline algorithm, indicating that the proposed GPIC algorithm can guarantee the accuracy of the original independent cascade process.

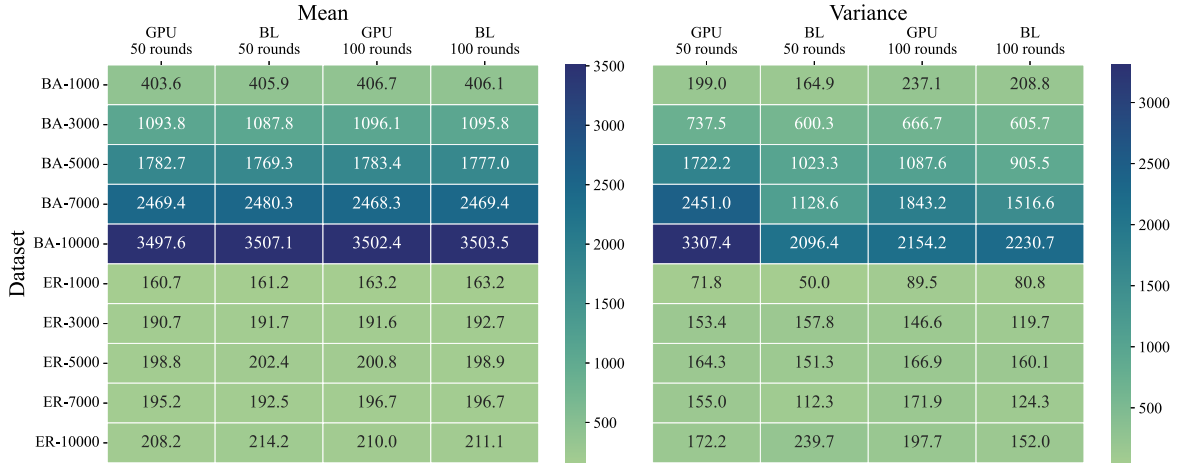



**Fig. 2.** Comparison of the spreading size between GPIC and baseline algorithm in Task 1. Seed size is fixed at 100.

In Fig. 3, taking the simulation results of a BA network with 5000 nodes as an example, we implement the spreading experiment of 1000 rounds of simulation by the GPIC algorithm and the baseline algorithm respectively. As shown in the figure, the GPIC algorithm has a mean spreading size of 1434.525 and a variance of 908.19, while the baseline algorithm has a mean of 1433.95 and a variance of 880.69. Perform a T-test on the two sets of results, and we get a T-value of 0.43 and a P-value of 0.67. It can be seen that the statistical difference between the two is very small and passes the T-test, validating that there is no

significant difference between the results from the GPIC and baseline algorithm. Figure 4 illustrates a slight difference in the variance of spreading size between the GPIC and the baseline algorithm within the BA network. This is because the seed node 0 connecting many nodes requires more rounds of simulation to reach stability.




**Fig. 3.** Comparison of the spreading size between GPIC and baseline algorithm in Task 2. Here 100 nodes with top degree are selected as seeds, and the immune seed sizes are 1%–20%; 500 and 1000 rounds of simulations are implemented respectively.

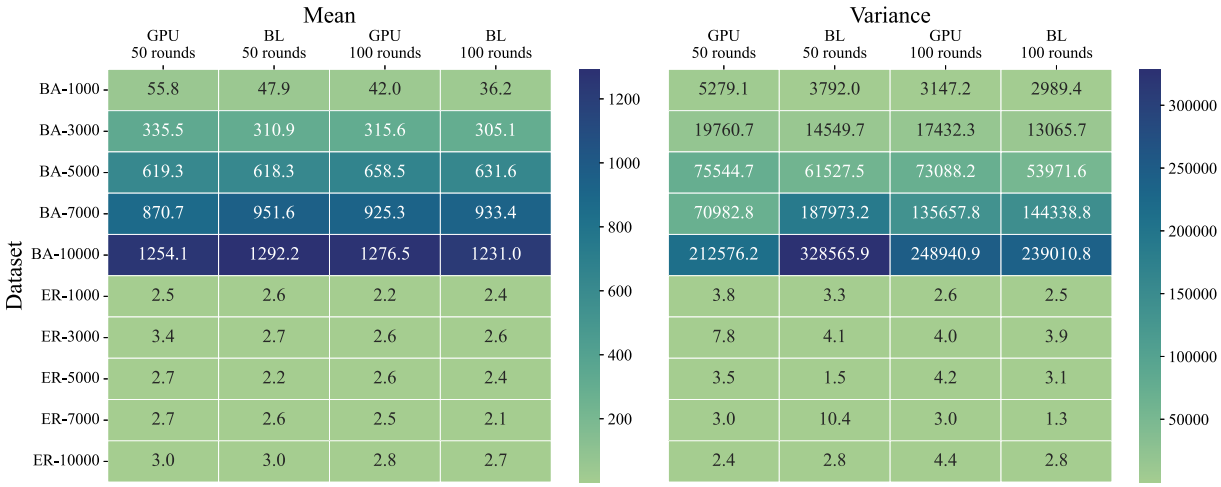



**Fig. 4.** Comparison of the spreading size between GPIC and baseline algorithm in Task 3.

Detailed results between the GPIC and baseline algorithm with different types of infection weights are presented in supplement material Tables A1–A3, and the results remain consistent.

### 4.3.2. Efficiency of GPIC

To analyze the acceleration effects of the proposed GPIC algorithm, we apply the GPIC and baseline algorithm to the three tasks mentioned above. As shown in Fig. 5(a), the execution time of the baseline algorithm increases proportionally with the number of simulation rounds, while the GPIC algorithm remains at a relatively low execution time regardless of the number of rounds. Furthermore, GPIC demonstrates efficiency across different network sizes, with only a slight increase in execution time as the network size grows.

In Fig. 5(b), GPIC achieves dozens of times acceleration performance in the BA network. In the ER network, since the size of seeds is fixed, the propagation of seeds is decreased with the increase of the network size, and the overall time consumption is reduced. Even so, the time consumption of the GPIC algorithm is still much lower than the baseline algorithm. Furthermore, in a network with a large scale, the overall execution time realized by the GPIC is reduced from thousands of seconds to tens of seconds, achieving an acceleration effect of two orders of magnitude.

Figure 6 presents time efficiency improvement of GPIC over baseline algorithm in three tasks. It shows that for Task 1 and Task 2, the GPIC can achieve hundreds of times acceleration effects when the network size is small, and tens of times improvement when the network grows larger. And for Task 3, the larger the network size, the greater the acceleration effect. This is because the number of simulated tasks is equal to the number of nodes.

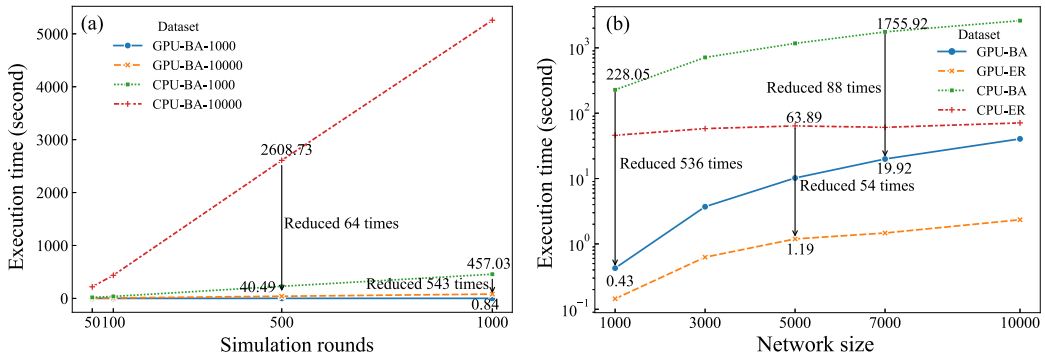



**Fig. 5.** Comparison of execution time of GPIC and baseline algorithm across different simulation rounds and networks sizes. (a) The time consumption of GPIC and baseline algorithm in Task 1, where BA networks vary in size. (b) The time consumption of GPIC and baseline algorithm in Task 2, where 20% of the nodes are immune and 500 rounds of simulation are considered.

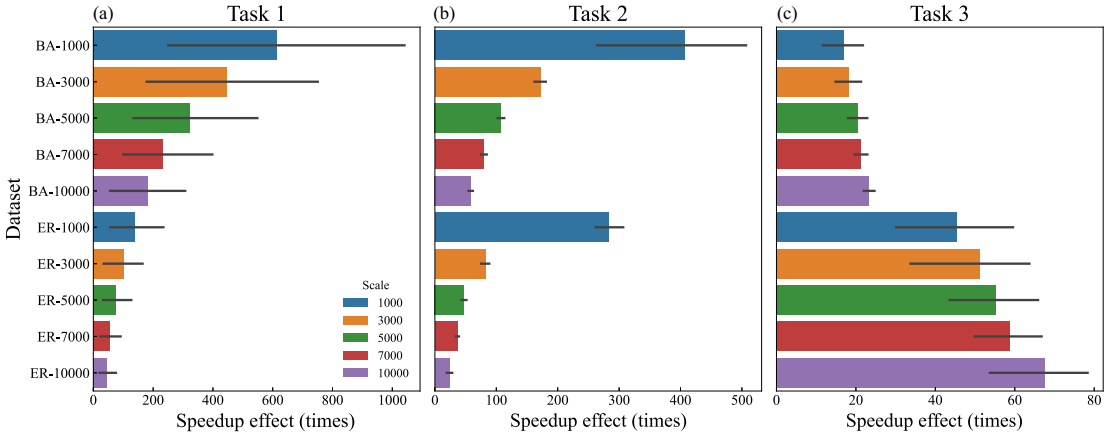



**Fig. 6.** Speedup effect of GPIC over baseline algorithm in three tasks. It shows the speedup effect of the BA and ER network at different scales, with the same scale represented by the same color.

### 4.3.3. GPIC performance on various platforms

We implement the GPIC algorithm on three different platforms to examine how hardware environment affects its performance. The platforms are equipped with MX150, GTX 1660SP, and RTX 3090, respectively. The three types of hardware are roughly compared in terms of CUDA core count, memory, and bandwidth, with the GTX1660SP having approximately 2–3 times the capabilities of the MX150, and the RTX 3090 showing about 20-30 times the performance metrics of the MX150.

We implement 50, 100, 500, and 1000 rounds of simulation on the BA network, and the experimental results show that compared with the CPU platform, using low-level graphics cards such as MX150 can also achieve an acceleration of 2–554 times, and using medium and high-level graphics cards can achieve a 21–1586 times acceleration effect (see supplement material Tables A4–A6). The performance of GPIC across MX150, GTX 1660SP, and RTX 3090 is illustrated in Fig. 7. Notably, GPIC's performance on the RTX 3090 significantly surpasses that on the MX150, while remaining comparable to the GTX 1660SP. This indicates that the GPIC algorithm can achieve good acceleration effects on consumer-grade graphics cards, and there is still a huge space for improvement on high-level graphics cards.

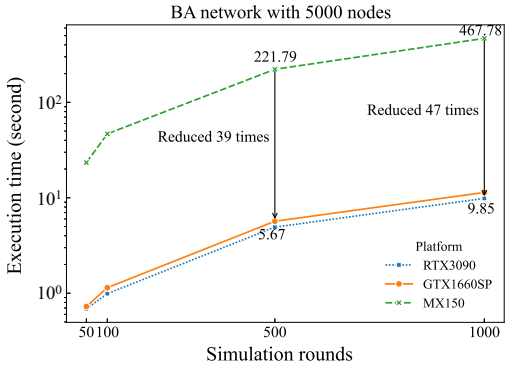



**Fig. 7.** GPIC's performance on various platforms with different types of hardware: MX150, GTX 1660SP, and RTX 3090.

### 4.3.4. GPIC test with real-world datasets

In recent years, with the continuous improvement of data collection capabilities, the scale of networks studied by scholars has expanded to the millions level. Large-scale network data has posed new challenges to the efficiency of simulations. The GPIC algorithm, based on the hardware conditions of a single computing node, has certain limitations and adaptability in implementing propagation simulations. We select four real-world datasets[36–38] with large scales and conduct testing tasks of 100 repeated simulations. The results demonstrate the computational capability of the GPIC algorithm on large-scale network datasets, as shown in Table 3. Tasks 1–3 indicate the

execution time of the GPIC algorithm for performing Tasks 1–3, respectively. The other headers are the same as those in Table 1.

For Task 1 and Task 2, we increased the number of Blocks to fully utilize more GPU cores during computation, thereby enhancing computational efficiency. However, Task 3 involves the need for intermediate result caching of size $N \times N$. For networks with hundreds of thousands of nodes, the required cache space exceeds RTX3090 memory (24 GB), which is beyond the capacity of ordinary hardware. Nevertheless, the GPIC algorithm benefits from the scheduling techniques of GPU-shared memory and can still perform computations using CPU memory. However, the frequent transfer between CPU memory and GPU memory limits the acceleration effect. Additionally, since each task in Task 3 is independent, it can be divided into multiple batches for computation. By controlling the GPU memory required for each batch to be within the capacity of the GPU's dedicated memory, the computational efficiency can also be effectively improved.

**Table 3.** Time consumption of the GPIC algorithm with real-world datasets in Tasks 1–3.

| Dataset | Nodes | Edges | $\langle k \rangle$ ($k^*$) | $c$ | Task 1 | Task 2 | Task 3 |
|---|---|---|---|---|---|---|---|
| Email-EUall[36] | 265214 | 365570 | 2.75(7636) | 0.455 | 29.51 | 92.89 | 623.73 |
| Web-BerkStan[37] | 685231 | 6649470 | 19.41(84230) | 0.634 | 564.96 | 90.71 | 25474.39 |
| Web-Google[37] | 916428 | 4322051 | 9.43(6332) | 0.624 | 312.02 | 16.70 | 11505.04 |
| YouTube[38] | 1157828 | 2987624 | 5.16(28754) | 0.172 | 213.78 | 5080.25 | 13957.49 |

## 5. Application of GPIC

In the field of information dissemination research, the IC model is employed for Monte Carlo simulations to estimate the spread size of information, which typically necessitates conducting multiple rounds of repeated experiments ($R$ times) and performing statistical analysis on the results. While it is generally recommended to conduct 100 to 200 rounds of simulation to balance costs and benefits, the exact number of repetitions required to achieve significantly approximate optimal results remains unclear.

Given the high efficiency of our GPIC algorithm, this section aims to (1) utilize the GPIC algorithm to perform a large number of repeated experiments to explore the impact of different repetition counts on the stability of simulation results; and (2) investigate the number of repetitions needed to achieve relatively stable simulation outcomes across various network scales.

### 5.1. Monte Carlo simulation

Here, we employ multiple BA scale-free network datasets with typical social network characteristics for the Monte Carlo simulation experiment, and conduct experiments on a system equipped with a high-performance GPU, specifically the Nvidia RTX3090.

In this experiment, when the statistical measures (such as variance and mean) of the simulation results show little fluctuation as the number of simulation rounds increases, we consider the current results to be stable and record the number of simulation rounds.

### 5.2. Simulation results

We construct a continuous sequence of data from the results of multiple rounds of simulation, recalculating the mean and variance of the sequence data with each additional data point, forming a continuously changing sequence of mean and variance. The trend and pattern of changes can be observed visually.

As shown in Fig. 8, before reaching 100 rounds, both the mean and variance exhibit significant fluctuations, indicating that the results are not yet stable. As the number of simulation rounds increases, particularly between 100 and 200 rounds, the mean and variance begin to stabilize, suggesting that the dynamic characteristics of the spreading process are diminishing. After approximately 500 rounds, both the mean and variance reach a relatively stable state, achieving full stability around 1000 rounds.

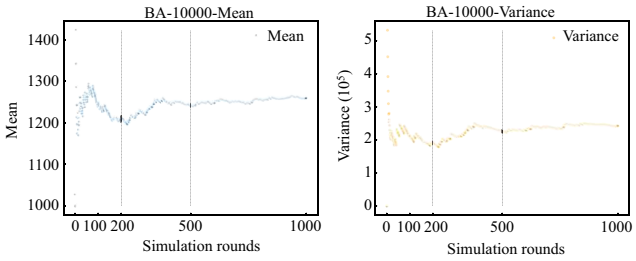



**Fig. 8.** The correlation between rounds of simulation and spreading size (mean and variance) of a single seed.

This indicates that for studies with higher costs, performing 100 to 200 rounds of simulation is acceptable, while for studies that require lower costs, it is advisable to conduct simulations for 500 rounds to ensure the reliability of the results. This approach allows for a balance between different research needs and resource constraints.

Furthermore, we also implement the same experiments in ER networks and homogeneous propagation type. Detail information is presented in supplement material Figs. S1–S3, and the results also remain consistent.

## 6. Conclusion and perspectives

This study proposes a GPU-based parallel algorithm to implement the independent cascade model, with the aim to solve the efficiency bottleneck problem faced by traditional algorithms when using the IC model with Monte Carlo simulation.

By designing new network data structures and efficient memory access patterns, as well as decomposing and paralleling tasks, the proposed algorithm significantly improves the efficiency and scalability of the IC model. The main contributions of this study include:

Algorithmic innovation: We propose a GPU-based parallel algorithm to implement the IC model, achieving high efficiency through GPUs' high parallel processing capabilities.

Performance improvement: We have proved that the algorithm can improve computational efficiency compared with baseline experiments, including datasets with different scales and rounds of simulations.

Significant application value: We have determined the optimal number of simulation rounds under limited computing power, providing practical insights for related research.

The GPIC algorithm holds extensive potential applications across various fields, extending beyond the scope of traditional information dissemination research. It can be utilized to analyze the spread of trending topics on social media, evaluate the effectiveness of advertising campaigns, and assess strategies for public opinion guidance in large-scale social networks. Additionally, it can rapidly validate the effectiveness of epidemic models in the context of epidemic spread simulation. The GPIC algorithm is also valuable for evaluating algorithms related to spreading dynamics and identifying vital nodes within networks.[39] Furthermore, its versatility allows it to be applied in interdisciplinary spreading research, such as in economics, ecosystems, and cybersecurity, where spreading dynamics play a crucial role.

The proposed GPU-based parallel IC algorithm has achieved remarkable results in improving simulation efficiency, but there are still some limitations that need to be improved in the future.

1) **Optimizing parameter setting** The current GPIC algorithm sets a fixed parameter value for the batch size involved in task decomposition. However, this value should be adjusted based on the hardware parameters of the GPU and the specific tasks at hand. Therefore, the GPIC algorithm can be optimized in the future to adaptively calculate more optimal parameter values according to different scenarios.

2) **Enhancing multi-GPU support** The current algorithm is mainly designed and optimized for a single GPU, and it can be optimized to extend to a multi-machine and multi-GPU computing environment, enabling parallel simulation on large-scale networks.

3) **Expanding to diverse propagation models** The current algorithm is designed for the independent cascade model, and it is expected to be expanded to more propagation models such as SI, SIR, and SIS,[40] therefore performing more extensive spreading simulations.

4) **Broadening practical applications** Our algorithm can be applied to other simulation scenarios, such as virus spreading simulation,[41,42] public opinion dissemination, etc., to verify its practicality and generalization ability.

In conclusion, this study demonstrates the feasibility of a GPU-based parallel algorithm on social network information simulation. It improves the efficiency of the IC model tremendously compared with the traditional algorithm. With the development of computing technology and the growth of network size, GPIC will provide an important tool and reference for research in social network fields.

## Program availability

The source code is openly available in the Science Data Bank at https://doi.org/10.57760/sciencedb.j00113.00183.

## Acknowledgements

The author acknowledges support from the National Natural Science Foundation of China (Grant No. T2293771), the STI 2030-Major Projects (Grant No. 2022ZD0211400), and the Sichuan Province Outstanding Young Scientists Foundation (Grant No. 2023NSFSC1919).

## References

[1] Barabási A L 2013 *Phil. Trans. R. Soc. A* **371** 20120375
[2] Newman M 2018 *Networks* (New York: Oxford University Press)
[3] Luding S 2005 *Nature* **435** 159
[4] Goldenberg J, Libai B and Muller E 2001 *Market. Lett.* **12** 211
[5] Goldenberg J, Libai B and Muller E 2001 *Acad. Market. Sci. Rev.* **9** 1
[6] Chen D, Lü L, Shang M S, Zhang Y C and Zhou T 2012 *Physica A* **391** 1777
[7] Lü L, Chen D, Ren X L, Zhang Q M, Zhang Y C and Zhou T 2016 *Phys. Rep.* **650** 1
[8] Malliaros F D, Rossi M E G and Vazirgiannis M 2016 *Sci. Rep.* **6** 19307
[9] Granovetter M S 1973 *Am. J. Sociol.* **78** 1360
[10] Granovetter M 1983 *Sociol. Theor.* **1** 201
[11] Lü L and Zhou T 2011 *Physica A* **390** 1150
[12] Wiki 2024 GPU Wikipedia
[13] Krizhevsky A, Sutskever I and Hinton G E 2017 *Communications of the ACM* **60** 84
[14] Huang Y P, Xia Y, Yang L, Wei J, Yang Y I and Gao Y Q 2022 *Chin. J. Chem.* **40** 160
[15] Wang B, Wald I, Morrical N, Usher W, Mu L, Thompson K and Hughes R 2022 *Comput. Phys. Commun.* **271** 108221
[16] Zhang Y H, Zhu H Z, Dong Y J, Zeng J, Han X P, Bratchenko I A, Zhang F R, Xu S Y and Wang S 2023 *Chin. Phys. B* **32** 118702
[17] Zhang B D, Tang Y H, Wu J J and Li X 2011 *Chin. Phys. B* **20** 098901
[18] Du Nguyen H A, Al-Ars Z, Smaragdos G and Strydis C 2015 *Design, Automation & Test in Europe Conference & Exhibition (DATE)* (Grenoble, France) p. 974
[19] Gao M, Li Z, Li R, Cui C, Chen X, Ye B, Li Y, Gu W, Gong Q, Wang X and Chen Y 2023 *Patterns* **4** 100839
[20] Zheng Z, Shi X and Jin H 2023 *IEEE Transactions on Big Data* **9** 677
[21] Liu X, Li M, Li S, Peng S, Liao X and Lu X 2014 *IEEE Transactions on Parallel and Distributed Systems* **25** 136

[22] Zou P, Lü Y s, Wu L d, Chen L l and Yao Y p 2013 *Simulation* **89** 1154
[23] Barabási A L and Albert R 1999 *Science* **286** 509
[24] Erdős P and Rényi A 1960 *Publ. Math. Inst. Hung. Acad. Sci.* **5** 17
[25] Eggemann N and Noble S D 2011 *Discrete Appl. Math.* **159** 953
[26] Watts D J and Strogatz S H 1998 *Nature* **393** 440
[27] Liu J G, Lin J H, Guo Q and Zhou T 2016 *Sci. Rep.* **6** 21380
[28] Kempe D, Kleinberg J and Tardos É 2003 *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Washington, D.C, USA) p. 137
[29] Kitsak M, Gallos L K, Havlin S, Liljeros F, Muchnik L, Stanley H E and Makse H A 2010 *Nat. Phys.* **6** 888
[30] Iannelli F, Mariani M S and Sokolov I M 2018 *Phys. Rev. E* **98** 062302
[31] Awerbuch B, Goldberg A V, Luby M and Plotkin S A 1989 *30th Annual Symposium on Foundations of Computer Science* (Research Triangle Park, NC, USA) p. 364
[32] Ren X L, Gleinig N, Helbing D and Antulov-Fantulin N 2019 *Proc. Natl. Acad. Sci. USA* **116** 6554
[33] Jajodia S, Noel S and O'berry B 2005 *Topological analysis of network attack vulnerability* (Boston, MA: Springer)
[34] Katz E and Lazarsfeld P F 1955 *Personal influence: The part played by people in the flow of mass communications* (New York: Routledge)
[35] Watts D J and Dodds P S 2007 *J. Consum. Res.* **34** 441
[36] Leskovec J, Kleinberg J and Faloutsos C 2007 *ACM Trans. Knowl. Discov. Data* **1** 2
[37] Leskovec J, Lang K J, Dasgupta A and Mahoney M W 2009 *Internet Mathematics* **6** 29
[38] Yang J and Leskovec J 2012 *IEEE 12th International Conference on Data Mining* (Brussels, Belgium) p. 745
[39] Yang P L, Zhao L J, Dong C, Xu G Q and Zhou L X 2023 *Chin. Phys. B* **32** 058901
[40] Keeling M J and Rohani P 2008 *Modeling Infectious Diseases in Humans and Animals* (Princeton: Princeton University Press)
[41] Liu S, Perra N, Karsai M and Vespignani A 2014 *Phys. Rev. Lett.* **112** 118702
[42] Pastor-Satorras R, Castellano C, Van Mieghem P and Vespignani A 2015 *Rev. Mod. Phys.* **87** 925