RESEARCH ARTICLE

Autonomous and ubiquitous in-node learning algorithms of active directed graphs and its storage behavior

Hui Wei*, Fushun Li, Weihua Miao

Laboratory of Algorithms for Cognitive Models, School of Computer Science, Fudan University, Shanghai, China

* weihui@fudan.edu.cn

OPEN ACCESS

Citation: Wei H, Li F, Miao W (2024) Autonomous and ubiquitous in-node learning algorithms of active directed graphs and its storage behavior. PLOS Complex Syst 1(3): e0000019. https://doi.org/10.1371/journal.pcsy.0000019

Editor: Jaya Sreevalsan-Nair, International Institute of Information Technology Banggalore, INDIA

Received: March 21, 2024

Accepted: September 11, 2024

Published: November 5, 2024

Peer Review History: PLOS recognizes the benefits of transparency in the peer review process; therefore, we enable the publication of all of the content of peer review and author responses alongside final, published articles. The editorial history of this article is available here: https://doi.org/10.1371/journal.pcsy.0000019

Data Availability Statement: All data are in the manuscript and/or supporting information files.

Funding: This work was supported by the NSFC Project (Project No. 61771146).

Abstract

The brain's memory system is extraordinarily complex, evidenced by the multitude of neurons involved and the intricate electrochemical activities within them, as well as the complex interactions among neurons. Memory research spans various levels, from cellular and molecular to cognitive behavioral studies, each with its own focus, making it challenging to fully describe the memory mechanism. Many details of how biological neuronal networks encode, store, and retrieve information remain unknown. In this study, we model biological neuronal networks as active directed graphs, where each node is self-adaptive and relies on local information for decision-making. To explore how these networks implement memory mechanisms, we propose a parallel distributed information access algorithm based on the node scale of the active directed graph. Here, subgraphs are seen as the physical realization of the information stored in the active directed graph. Unlike traditional algorithms with global perspectives, our algorithm emphasizes global node collaboration in resource utilization through local perspectives. While it may not achieve the global optimum like a global-view algorithm, it offers superior robustness, concurrency, decentralization, and biological feasibility. We also tested network capacity, fault tolerance, and robustness, finding that the algorithm performs better in sparser network structures.

Author summary

In this paper, we delve into how biological neuronal networks encode, store, and retrieve information, aiming to model the brain's memory system and propose practical algorithms for memory characterization and information storage at the algorithmic level. To characterize memory effectively, we must first identify its physical counterpart. We abstract the biological neuron network as an active directed graph, which serves as the framework for memory storage. According to the theory of memory engram and synaptic plasticity, memory is the co-activation of specific neuronal clusters and synaptic sets, which is reflected in the directed graph as the co-activation of specific point sets and edge sets. These activated point sets and edge sets are actually a connected subgraph of the whole active directed graph. Therefore, we propose to consider this connected subgraph

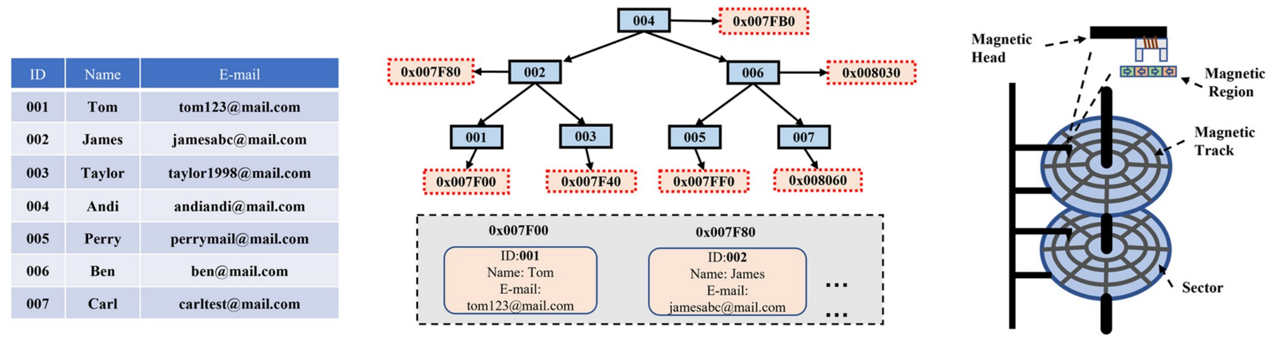
Competing interests: The authors have declared that no competing interests exist.

as the physical counterpart of memory. We design a parallel distributed access algorithm based on the scale of the directed graph to explore whether this assumption meets the properties of stability, distinguishability, less interference, and incrementalism exhibited by memory. Our approach offers a more biologically realistic network model, focusing on the impact of connections between neurons and structure on memory rather than numerical characterization.

1 Introduction

Memory is one of the brain's most crucial and intricate cognitive abilities. It serves as the foundation for information processing and supports the development of higher cognitive functions such as thinking, learning, and decision-making. Neurobiology researchers have investigated memory mechanisms at various levels, ranging from molecular and cellular to brain slice and whole-animal studies [1]. Neuroimaging techniques like functional magnetic resonance imaging (fMRI) and positron emission tomography (PET) enable the exploration of the brain's functional connectivity and neural activity patterns during memory tasks. Meanwhile, the processes of memory encoding, storage, and retrieval are studied using neurophysiological techniques such as single-cell recordings and electroencephalography (EEG), along with behavioral cognitive tasks. However, each level of memory research has its focus and limitations, providing only a partial account of the brain's memory mechanisms. The brain continuously receives sensory information from the environment, encoding it into various variables. These variables are transmitted through neural circuits that encode and store key information. Despite advancements, many details of how biological neuronal networks encode, store, and retrieve information remain unknown. Data from cognitive-behavioral tests, neuroimaging, and electrophysiological experiments offer valuable but limited insights.

To draw an analogy between the human brain's memory system and a computer storage system, the implementation process of computational storage is well-understood, proceeding clearly from the bottom to the top. However, much remains unknown about the implementation process of brain memory. For example, consider a database management system (DBMS), which is a widely used data management and storage system found in almost every computer. Through a DBMS, users can easily access and update data. Take the classic relational database MySQL as an example. When using MySQL to store information, the DBMS records the data in the form of a B+ tree into a file. In this process, the DBMS first receives SQL statements from the user for storing information. These statements are parsed, optimized, and executed by the database engine. The data storage process involves creating and writing files. Using the Linux file system as an example, the operating system first finds an unallocated inode and records basic attributes such as file size, owner, and creation time in it. It then finds a number of unallocated data blocks based on the data size and writes the binary encoded data. When data needs to be written to the disk, the operating system passes the logical address of the data to the disk. The disk's control circuitry translates the logical address into a physical address, determining which track and sector the data to be read is on. It then moves the head to the corresponding track, rotates the target sector under the head, and uses an electric current to generate a strong magnetic field at the head to change the polarity of the magnetic particles passing over the disk, enabling the writing of data. As illustrated in Fig 1, from encoding to storing data, the details of each of these processing steps and the interface between them are well-defined. While it is clear that a human memory system similar to a DBMS can perform the same task, based on our current knowledge of neuroscience, psychology, and



ID	Name	E-mail
001	Tom	tom123@mail.com
002	James	jamesabc@mail.com
003	Taylor	taylor1998@mail.com
004	Andi	andiandi@mail.com
005	Perry	perrymail@mail.com
006	Ben	ben@mail.com
007	Carl	carltest@mail.com

(a) Logical View: Relational Data Sheet (b) B-Tree Data Management (c) Pyhsical View: Hard Disk

Fig 1. Using database management systems as an analogy, the brain's memory system lacks such a clear and complete chain of signal processing.

https://doi.org/10.1371/journal.pcsy.0000019.g001

computational modeling, we are unable to provide a logically consistent explanation of the complete mechanism of memory implementation as detailed as that of a DBMS, from the bottom layer of impulse issuance to the top layer of memory function realization.

Directed graphs, a classical data structure in computer science, are often employed to depict connections between information. Biological neuronal networks are readily depicted as directed graphs of elements and their connections [2, 3]. In these networks, neurons receive and integrate stimulus signals from upstream neurons via dendrites and the cell body, subsequently transmitting stimulus signals to downstream neurons via axons. Neurons can also self-regulate through synaptic plasticity mechanisms such as Spike-Timing-Dependent Plasticity (STDP) [4]. STDP is a physiological process that adjusts the strength of neural connections in the brain, based on the relative timing of action potential inputs and outputs. If an input signal from a neuron consistently occurs just before the neuron's output signal, the connection becomes stronger. Conversely, if the input signal occurs after the neuron's output, the connection weakens. It's evident that there is no super-manager with a global vision controlling the behavior of every neuron or node. A single biological neuron is a complex computational unit involving numerous state variables, such as nerve voltage [5], synaptic activation [6], synaptic strength [7], synaptic connectivity [8], phosphorylation level [9], mRNA concentration [10], transcriptional regulation [11], and neuromodulatory signals [12]. In biological neural networks, different subtypes of neurons exhibit distinct morphological and electrophysiological characteristics. Therefore, neurons cannot be simplified into nodes that are merely represented numerically in a directed graph. Static directed graphs only fulfill real-time recording functions, with the nodes usually only characterizing values. It is more accurate to abstract biological neuronal networks as active directed graphs. Viewing the active directed graph from a multi-agent system perspective, each node can be seen as an independent and fully autonomous agent [13]. Each node's scope is limited to its upstream and downstream connected nodes, with information passed between neighboring nodes via directed edges. Every decision made by a node is based on local information and can be continuously learned and optimized. The intrinsic behavior of each node and its external upstream and downstream connections are distinct, resulting in the entire graph displaying rich and complex dynamical behavior. This raises a question for memory studies: how do active directed graphs with neurobiological constraints such as local horizons, autonomous mobility, and self-adaptation store, encode, and retrieve information?

In neurobiology, a memory is any change in the activity or connectivity of the nervous system triggered by a stimulus or a brain state that lasts longer than the triggering event. It has been proposed that memories are encoded in an engram complex, yet many questions remain unanswered. These include how information is stored in the imprint, how the structure of the imprint impacts memory quality, strength, and precision, and how multiple imprints interact with each other [14]. Viewed from the perspective of an active directed graph, such a graph consists of numerous nodes and directed edges. The permutations of these nodes and edges can generate a vast array of connected subgraphs. If these connected subgraphs are seen as a resource, they can be understood to characterize states where nodes are active or related to each other. Consequently, the number of states that can be represented by the entire directed graph is substantial, allowing it to store or remember a significant amount of content. In an active directed graph where each node exhibits autonomous and independent behavior, making decisions based solely on locally obtained information, the shape and number of functional connected subgraphs are unpredictable. Furthermore, if we introduce the possibility of incremental, adaptive learning at each node, the diversity and number of functional subgraphs in an active directed graph become even greater. To explore the brain's memory mechanisms within an active directed graph framework, it is essential to establish neurobiologically consistent behavioral guidelines for the nodes.

How to form functional connected subgraphs, consolidate them, effectively utilize limited node and path resources, and ensure compatibility or minimal interference among multiple functional connected subgraphs are key topics investigated in this paper. The primary focus is on realizing storage in a directed graph where every node is dynamic, and how to freely form, consolidate, and activate functional connected subgraphs. The main contribution of this paper is the proposal of a parallel distributed storage algorithm based on node scales in an active directed graph. Unlike traditional algorithms that rely on a global field of view, the design challenge here is that nodes must achieve global collaboration on resource usage through their very limited local field of view. While this approach may not always achieve the global optimum like algorithms with a global field of view, it offers superior robustness, concurrency, decentralization, and biological feasibility.

Directed graphs have various application modes in storage, and one common mode is the abstract modeling of neuronal networks. One such example is the Hopfield network proposed by John Hopfield in 1982 [15]. It is a fully connected binary recurrent neural network that characterizes the network state by an energy function. Each iteration of the network proceeds towards energy reduction until it reaches a steady state, also known as an attractor. The number of attractors represents the network capacity, which is approximately 0.14N, where N represents the number of nodes in the network. When implementing the associative memory function, the Hopfield network enables complete content retrieval by only part of the sample. However, the capacity of the Hopfield network increases linearly with its network size, making it difficult to preserve too many samples. In 2016, Krotov and Hopfield introduced the discrete modern Hopfield network [16], which allows network capacity to be extended by changing the network energy function and the update rule, but at the corresponding cost of requiring a large number of hidden layer nodes. Demircigil et al. [17] further extended the energy function by introducing exponential interaction functions, increasing the network capacity. In 2021, Ramsauer, Hubert, et al. [18] extended the energy function of modern Hopfield networks from discrete to continuous states while maintaining exponential storage capacity and fast convergence. Hopfield networks, as classical self-associative computational models, enable mapping between vectors of the same dimension. The bidirectional associative memory (BAM) model proposed by Bart Kosko in 1988 [19] can realize both self-association and hetero-association, i.e., mapping between vectors of different dimensions. The model comprises two layers of neurons connected by a weight matrix, which encodes the mapping relationships of all samples. Activating any layer of neurons and iterating through the network results in a correlated output in the neuron on the other layer. In 2021, Bart Kosko [20] introduced a bidirectional backpropagation algorithm in BAM to update matrix parameters dynamically. The original structure was also extended to have any number of hidden layers. The network capacity is increased. In this mode, directed graphs simulate the structure of biological neuron networks, and the network structure is often fixed, such as fully connected or hierarchically connected. The impact of structural parameters such as connectivity, clustering coefficient, and average path length on network performance is not considered. The implementation of the storage function relies more on the weight parameters and update rules of the network, which remains the weight-centric theory. Moreover, all storage contents need to be determined in advance, and the weights are calculated and written at once, making them hard to update incrementally or partially. The local damage to the network will affect the global network, and the scalability of the network scale is not good. In recent research, Wei et al. [21] abstracted biological neural networks into active directed graphs, exploring how directed graphs at the cellular level can encode, store, and retrieve information. This research, grounded in neurobiology, integrates theories from graph theory, multi-agent systems, and parallel distributed processing. It emulates the connectivity characteristics of biological cortical neural networks to construct directed graphs and designs a node-adaptive connectivity learning algorithm under the premise of finite resource competition. Wei et al.'s work provides an innovative and feasible perspective for exploring the memory mechanisms of the brain. Senk et al. [22] proposed a set of standardized connectivity concepts for neural network modeling, including deterministic and probabilistic connection rules, considering the impact of node distribution in metric space on connection probability. They found that many published neural network models have unclear or incomplete descriptions of connectivity. By introducing unified graphical symbols and descriptive language, Senk et al. aim to improve the clarity and reproducibility of model descriptions, promoting the shareability and reusability of computational neuroscience research. These advancements not only help us understand the working principles of biological neural networks more deeply but also provide a solid theoretical foundation for the design and optimization of neural networks.

In contrast to artificial neural networks that rely on fixed connection patterns, weight parameters, and update rules, or static directed graphs that primarily function for fact recording, this paper presents a method for storing information in a directed network in a distributed manner. This approach leverages the autonomous and dynamic behaviors of numerous nodes in the network, such as resource acquisition and competition. The information content is differentiated based on the distinct combinations of nodes and edges. Subgraphs serve as the information storage carriers without relying on super nodes. Additionally, there is no need for a global view. Information is stored through nodes and edges' local, limited, and adaptive dynamic behaviors. This subgraph-based computational storage model ensures that the stored information remains stable, distinguishable, and fault-tolerant. It also enables incremental storage of information. The performance of this storage method relies on the network's structural characteristics and the nodes' adaptive learning algorithm.

The brain's neuronal network is considered the physiological basis for information processing and mental representation. The brain is an extremely complex information processing system in terms of both function and structure, making the understanding of its memory mechanisms a longstanding challenge in memory research. Computational models of memory based on artificial neural networks are an approximate tool to enhance our understanding of the human brain's memory system. Despite significant progress in neural networks, these models do not closely resemble neurobiological aspects. To further comprehend the brain's information processing mechanisms, it is necessary to establish neural network models that are more consistent with biological realities based on neurobiological evidence.

Recent studies on engram cell ensembles have shown that the memory trace for a given memory is not necessarily located in a single anatomical location but is distributed across multiple locations connected by specific memory patterns, forming a memory engram. According to the engram hypothesis, memory information is believed to be stored in the network formed by these neurons. If we attempt to understand this from the perspective of directed graphs, neurons are represented as nodes, and synapses between neurons are represented as directed edges. Thus, a memory engram is a connected subgraph formed by activated nodes, with different connected subgraphs representing different memory contents.

In this paper, we abstract the brain's neuronal network into a directed graph and explore how active directed graphs with neurobiological constraints (such as local vision, autonomous mobility, and adaptability) store, encode, and retrieve information. A memory model based on directed graphs is an appropriate modeling method for cortical neuronal networks, exploring the memory mechanisms at the level of cortical neuronal networks from the perspective of directed graphs.

2 Materials and methods

2.1 Subgraph-based storage implementation

Let $G = (V, E)$ be a directed graph, where V denotes the set of nodes and E denotes the set of edges. If $G_{sg} = (V_{sg}, E_{sg})$ is a subgraph of G, then it follows that $V_{sg} \subseteq V$ and $E_{sg} \subseteq E$, denoted as $G_{sg} \subseteq G$. In a subgraph-based storage implementation, information is recorded in the form of a series of active nodes and interconnected pathways between them, i.e., a subgraph in the network. For instance, consider the message to be stored as: "While observing a red apple on a tree, I also saw a chirping robin." In this case, nodes representing semantic elements like red, circle, branch, and chirping are activated simultaneously and propagate stimulus along the directed edges in the network. These nodes are referred to as the initial nodes of this sample. During the stimulus propagation process, the initial nodes activate some otherwise inactive nodes, called communication nodes, which are crucial for establishing the pathways. Not all nodes are directly connected by edges in non-fully connected networks, so communication nodes serve as bridges to establish pathways between the initial nodes. These pathways represent associations between semantic elements, such as recalling a robin when seeing an apple again. This occurs when the initial node representing the apple is activated, and the stimulus is passed along the stored pathway in the network, finally activating the node representing the robin. This implementation draws inspiration from cognitive psychology studies on long-term memory [23, 24].

However, the initial idea is insufficient and requires the design of specific implementation details. For example, a node may only characterize a fundamental physical feature, necessitating multiple nodes to represent a concept like an apple. The initial node may activate some communication nodes, which may activate others. Fig 2A shows 30 randomly selected nodes as initial nodes in a directed network. Fig 2B shows a stable subgraph obtained by propagating the stimulus of these 30 initial nodes through the network with continuous iterations.

Another factor that makes subgraphs suitable for information storage is the vast number of potential subgraphs present in the network. Given $m = |E|$, there can be up to 2^m subgraphs in graph G. Thus, using subgraphs as information storage carriers is a promising idea. The challenge lies in ensuring that the subgraphs do not interfere with or confuse each other, effectively utilizing the node and edge resources of the entire directed graph, enabling incremental storage, reducing unfair resource occupation due to varying sample upload orders, and sharing resources among multiple samples. These technical aspects need to be solved by the parallel distributed network adaptive learning algorithm.

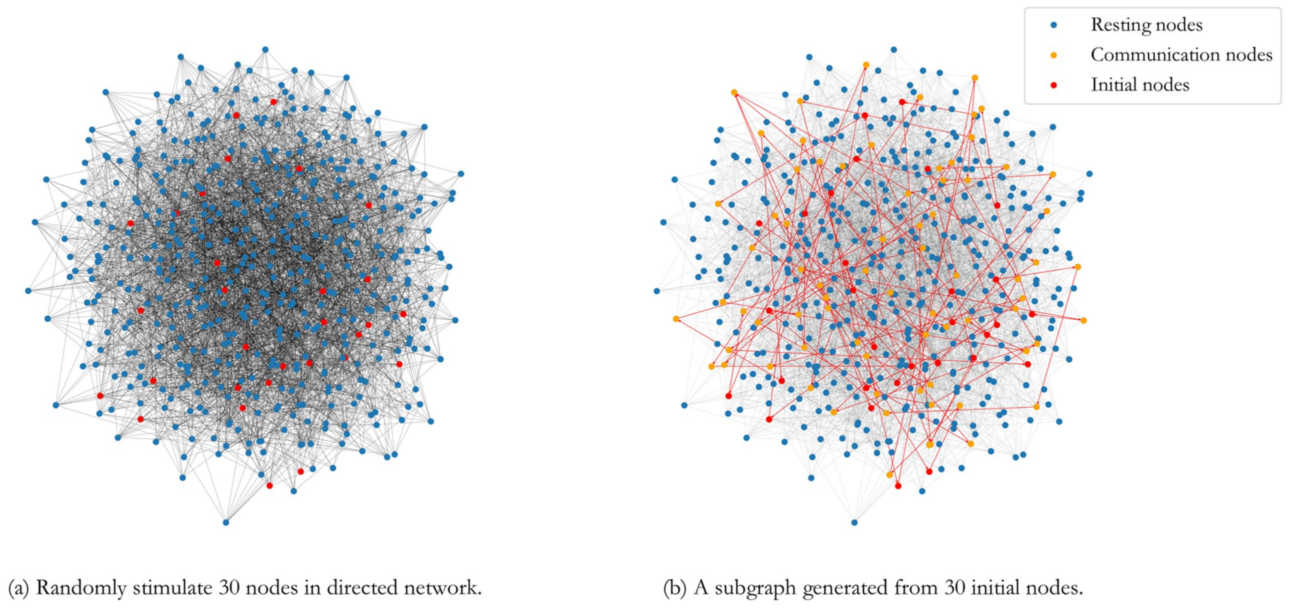

Resting nodes
Communication nodes
Initial nodes
(a) Randomly stimulate 30 nodes in directed network.
(b) A subgraph generated from 30 initial nodes.

Fig 2. A sample can be stored as a subgraph in a directed graph. (a) Random activation of 30 nodes in the network. (b) Activated nodes propagate stimulus in the network to form a subgraph.

https://doi.org/10.1371/journal.pcsy.0000019.g002

2.2 Subgraph generation, storage, and retrieval

The storage and retrieval of samples are processes of the initial nodes propagating the stimulus in the network and eventually forming a stable subgraph. Assuming the stimulus propagation time between nodes is constant. The subgraph eventually reaches a stable state through iterations. The formal definition of a stable subgraph is as follows: let V^t represent the set of all active nodes in the network at time t. There exists a minimum time t' such that $V^{t'} \neq V^{t'-1}$ and $V^{t'} = V^{t'+k}$ for any positive integer k. At this point, the network is considered to be in a stable state at time t', and the subgraph comprising all active nodes and edges is the stable subgraph. There are two primary concerns: first, how the subgraph is recorded in the network, and second, what rules nodes use for stimulus propagation. Section 2.2.1 describes the recording of subgraphs, while sections 2.2.2 to 2.2.4 outline the stimulus propagation rules. Section 2.2.5 presents the specific procedure for sample storage and retrieval.

2.2.1 The node internal index table records the local upstream and downstream connectivity traces. The storage of subgraph structures involves recording the connectivity paths between active nodes, which can only be accomplished by the nodes themselves based on their local perspectives. This necessitates that active nodes individually record activation information both upstream and downstream of themselves. Let the node be v_i, where i is the index of the node. Define the activation trace of node v_i as a path fragment consisting of the active fan-in nodes and active fan-out nodes of node v_i. Storing the activation traces of all active nodes during this sample storage process will completes the subgraph storage.

In this paper, we store node activation traces by introducing an index table in each node, a data structure with small capacity, easy access, and simple updating. Fig 3 shows the structure of the index table, which contains two columns: the first for active fan-in nodes and the second for active fan-out nodes. Fig 3A shows that node v_b's index table contains no content before storing the samples. Once a sample is stored, its corresponding activation trace is saved in its index table. As shown in Fig 3B, during sample retrieval, if node v_b receives the same or similar

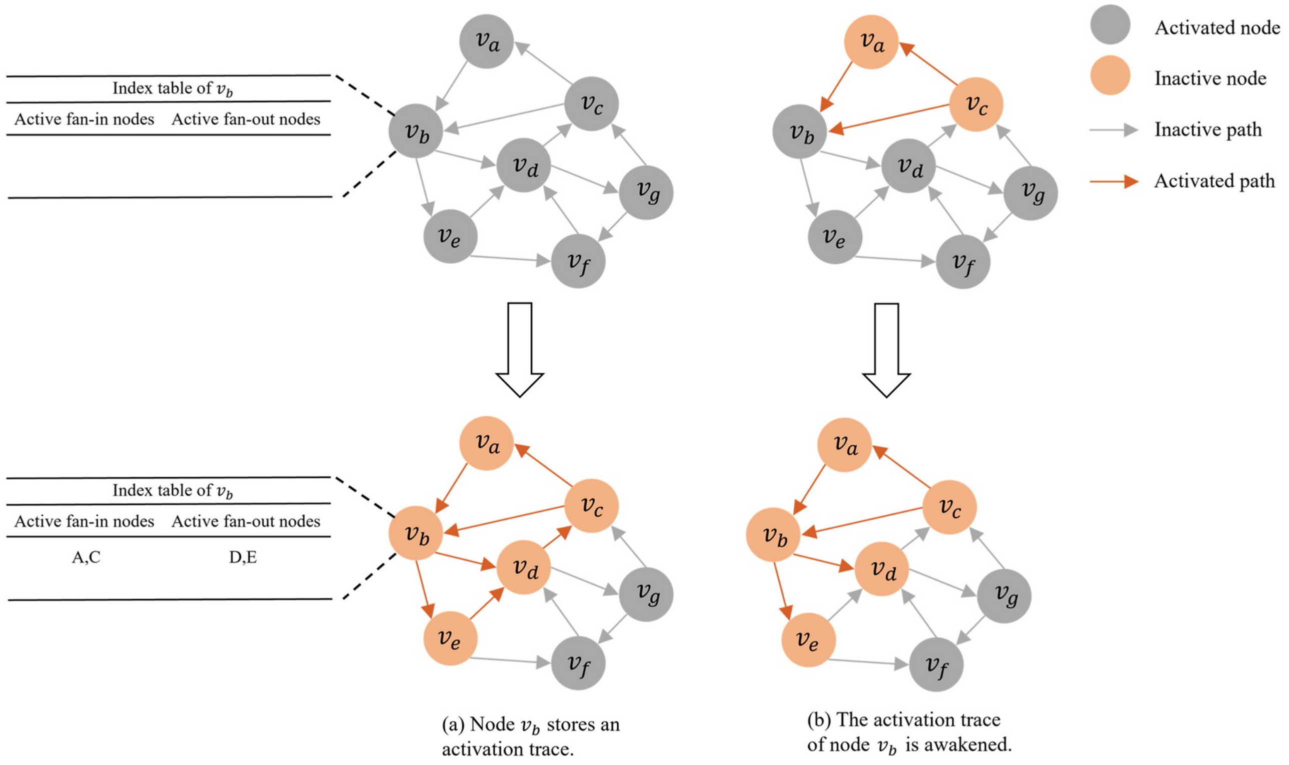


Fig 3. How to record activation traces within a single node. (a) After storing a sample, node v_b's index table adds an activation trace. (b) When node v_b receives the same or similar input again, it reuses the previously recorded activation traces.

https://doi.org/10.1371/journal.pcsy.0000019.g003

input as the recorded activation traces, it generates the corresponding output based on the historical records in the index table.

Defining an index table inside each node that records upstream and downstream active path pairing relationships may appear straightforward and crude, but it is also biologically feasible. Biological neurons have dendrites that receive inputs from multiple directions and axonal that transmit outputs in different directions, creating a many-to-many connection. Actual physical signaling between upstream and downstream neurons relies on synapses, regulated by combinations of diverse neurotransmitters and ion pumps. These mechanisms precisely control the direction and intensity of positive and negative charge flow. Additionally, differences in synapse location, such as being distal or proximal to the axonal, on dendrites or axons, or on the main pathway or terminal, can precisely control the activation and deactivation of specific action potential transmission pathways. In conclusion, this highly precise and diverse molecular-level and subcellular-level modulation and their combinations equip biological neurons with various pathway control mechanisms at the microcircuit level [25, 26]. As a result, a biological neuron can achieve diverse pathway control of signaling within its small neighborhood, relying on a complex set of electrochemical processes [27]. This has inspired the design of directed graph nodes' internal behavior, allowing them to function like network routers capable of differentially leading fan-outs based on fan-in variations. An index table with limited storage space is a simple, functional equivalent implementation.

2.2.2 Intra-node stimulus propagation algorithm. The creation of subgraphs depends on the propagation of stimulus between nodes. Stimulus propagation consists of two aspects: node activation rules, i.e., how nodes are activated, and stimulus propagation rules, i.e., determining the downstream nodes to which the stimulus is propagated. In this paper, the node activation rule employed is a fixed-probability activation model, where a node will be activated with a fixed probability upon receiving input. The node becomes active and begins delivering stimulus to downstream nodes if successfully activated. Two types of stimulus propagation rules are used in this paper: the first reuses similar historical activation traces, and the second employs a weighted random selection algorithm.

The specific process of stimulus propagation among nodes is as follows: each resting state node has a fixed probability, H, of being activated after receiving the stimulus. Once a node is activated, if its index table is empty, it randomly selects several downstream nodes with equal probability for stimulus delivery. If the index table is not empty, the similarity between the input and each item in the node index table is calculated first. In this paper, the F1 score [28] is used as a metric to evaluate the similarity of two node sequences. The F1 score is a statistical measure of the accuracy of a binary classification model, which is the harmonic mean of precision and recall. When comparing similarity, either one of the node sequences can be treated as the predicted value and the other as the actual value, and the corresponding F1 score is calculated. Higher scores indicate greater similarity. The corresponding historical activation traces are reused if the maximum similarity exceeds the threshold. Otherwise, a weighted random selection algorithm is used to select downstream nodes for stimulus delivery.

Algorithm 1: *NodeStimulusSpreading*(v_a)

```
Data: Table[$v_a$]: $v_a$'s index table. currentIn[$v_a$]: Current input of $v_a$.
      Threshold: Similarity threshold. H: Probability of being acti-
      vated. G[$v_a$]: $v_a$'s adjacency table. eOutNum: The expected value
      of output size. freq[$v_b$]: The occurrence frequency of $v_b$ in the
      index table of $v_a$.
Input: $v_a$: Current node
Output: aOut: The fan-out nodes of $v_a$
1 if randomValue(0, 1) > H then
    /* Activation failed                                          */
2   return NULL;
3 end
4 mxF1 ← 0;
5 aOut ← NULL;
6 for item in Table[$v_a$] do
    /* Traverse the index table and find the item most similar to cur-
       rentIn[$v_a$];                                               */
7   f1Score ← CalculateF1Score(item.in, currentIn[$v_a$])
8   if f1Score > Threshold and mxF1 < f1Score then
9     mxF1 ← f1Score;
10    aOut ← item.out;
11  end
12 end
13 if aOut ≠ NULL then
14   return aOut;
15 end
16 for $v_b$ in G[$v_a$] do
    /* Calculate the occurrence frequency of $v_b$ in the index table of
       $v_a$                                                        */
17   freq[$v_b$] ← getFrequencyOfOccurrence($v_b$, Table[$v_a$]);
18 end
    /* Weighted random selection algorithm                         */
19 aOut ← randomChoose(G[$v_a$], freq, eOutNum);
20 return aOut;
```

The weighted random selection algorithm is based on the frequency of the downstream node appearing in all active fan-out nodes in the current node index table. The higher the

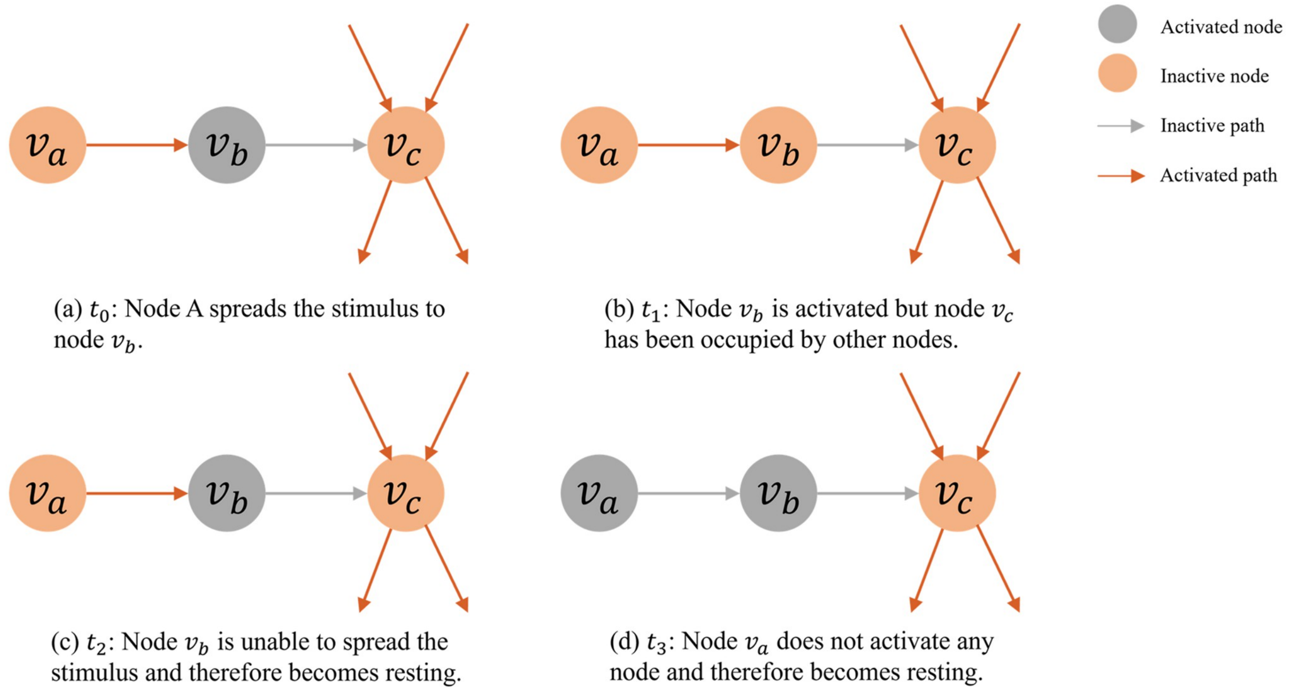

v_a
v_b
v_c
● Activated node
● Inactive node
Inactive path
Activated path
(a) t_0: Node A spreads the stimulus to node v_b.
(b) t_1: Node v_b is activated but node v_c has been occupied by other nodes.
(c) t_2: Node v_b is unable to spread the stimulus and therefore becomes resting.
(d) t_3: Node v_a does not activate any node and therefore becomes resting.

Fig 4. Node resource-grabbing rules. (a) Node v_a propagates stimulus to node v_b. (b) After node v_b is activated, it finds that other nodes have occupied node v_c. (c) Node v_b becomes a resting state as it cannot continue to propagate stimulus. (d) Node v_a becomes a resting state because it has not activated any downstream nodes.

https://doi.org/10.1371/journal.pcsy.0000019.g004

frequency of occurrence, the lower the chance of being selected. The introduction of this algorithm allows each active node to distribute stimulus evenly, thus maximizing network resource utilization. The algorithm pseudo-code is shown in Algorithm 1. The value of H affects the subgraph size. The larger H is, the more nodes participate in subgraph formation and the better the connectivity. However, the corresponding cost of network resources is also larger. In this paper, H is set at 60% for testing.

2.2.3 Node resource grabbing rules. Nodes are considered limited resources in a directed graph, adhering to the first-come, first-served preemption rule. Activating a node can be seen as the occupation of a node resource. When performing stimulus propagation, nodes can acquire the occupancy of downstream nodes to avoid passing stimulus to already occupied nodes. If an active node does not successfully activate any downstream nodes, it returns to a resting state. A change in the state of some active nodes may trigger a chain reaction that causes more active nodes to become resting. This situation is called the avalanche effect.

As shown in Fig 4, at t_0, node v_b receives a stimulus from node v_a, then subsequently activated at t_1. However, because node v_c has been occupied, node v_b cannot transmit the stimulus to node v_c, causing node v_b to revert to the resting state at the t_2 moment. At this point, node v_a is not activating any nodes due to the change in the state of node v_b. Therefore, at t_3, node v_a also becomes resting due to the avalanche effect.

2.2.4 Several problems are caused by insufficient resources in subgraph generation. As the sample count grows, new subgraphs may face resource shortages, hindering storage. Insufficient network resources generally fall into four categories:

1. The node index table has a capacity limit. When the number of samples stored in the network reaches a certain level, it becomes hard to store new samples.

2. The network is poorly connected, and the stimulus propagation rule carries a certain level of randomness, as well as the existence of the avalanche effect, which leads to an inability to establish a pathway between the initial nodes.

3. The samples already stored in the network interfere with the samples currently about to be stored. This is because nodes may reuse historical activation traces when they are activated. Although this is an optimization strategy to increase network capacity, it somewhat affects the storage of current samples.

4. Some active nodes take up too many node resources during the current activation, resulting in no resources available for other nodes.

For the first case, the capacity of the index table can be defined as the maximum number of output types that a node can store, as there may be many different inputs corresponding to the same output. The capacity can also be expanded by reasonably discarding and merging the contents of the index table. Specifically, when a node's index table capacity reaches its upper limit, the node will search for the two most similar activation traces to merge. The similarity here refers to the similarity of the active fan-in nodes in the two activation traces, and merging refers to taking the intersection of the active fan-out nodes of the two activation traces. If the differences between the activation traces are both large, the one with the lowest strength is discarded. The strength here refers to the number of samples that the activation trace has been involved in storing. The more involved, the higher the strength. By reasonably merging and discarding, it is possible to increase network capacity as much as possible at the expense of certain recall accuracy and completeness. The pseudo-code of the algorithm is given by Algorithm 2.

Algorithm 2: *ActivationTracesUpdating*()

```
Data: activeNodes: active nodes set. Outputs[$v_a$]: A collection of dif-
      ferent output in Table[$v_a$]. K: The upper limit of the output
      type.
for $v_a$ in activePoints do
  Table[$v_a$][in[$v_a$]] = out[$v_a$];
  if out[$v_a$] not in Output[$v_a$] then
    Output[$v_a$].add(out[$v_a$]);
  end
  if Output[$v_a$].size() > K then
    /* Find the two most similar outputs to merge */
    out1, out2 $\leftarrow$ findTwoMostSimilarOutput(Output[$v_a$]);
    if out1 $\neq$ NULL and out2 $\neq$ NULL then
      out3 $\leftarrow$ merge(out1, out2);
      Output[$v_a$].remove(out1);
      Output[$v_a$].remove(out2);
      Output[$v_a$].add(out3);
      /* Change the output of all items whose output is out1 or out2
         in Table[$v_a$] to out3 */
      changeItemsInTable(Table[$v_a$], out1, out3);
      changeItemsInTable(Table[$v_a$], out2, out3);
    else
      /* If not found, discard the output with the lowest strength */
      out1 $\leftarrow$ findLowestIntensityOutput(Table[$v_a$]);
      Output[$v_a$].remove(out1);
      /* Delete all items whose output is out1 */
      changeItemsInTable(Table[$v_a$], out1, NULL);
    end
  end
end
```

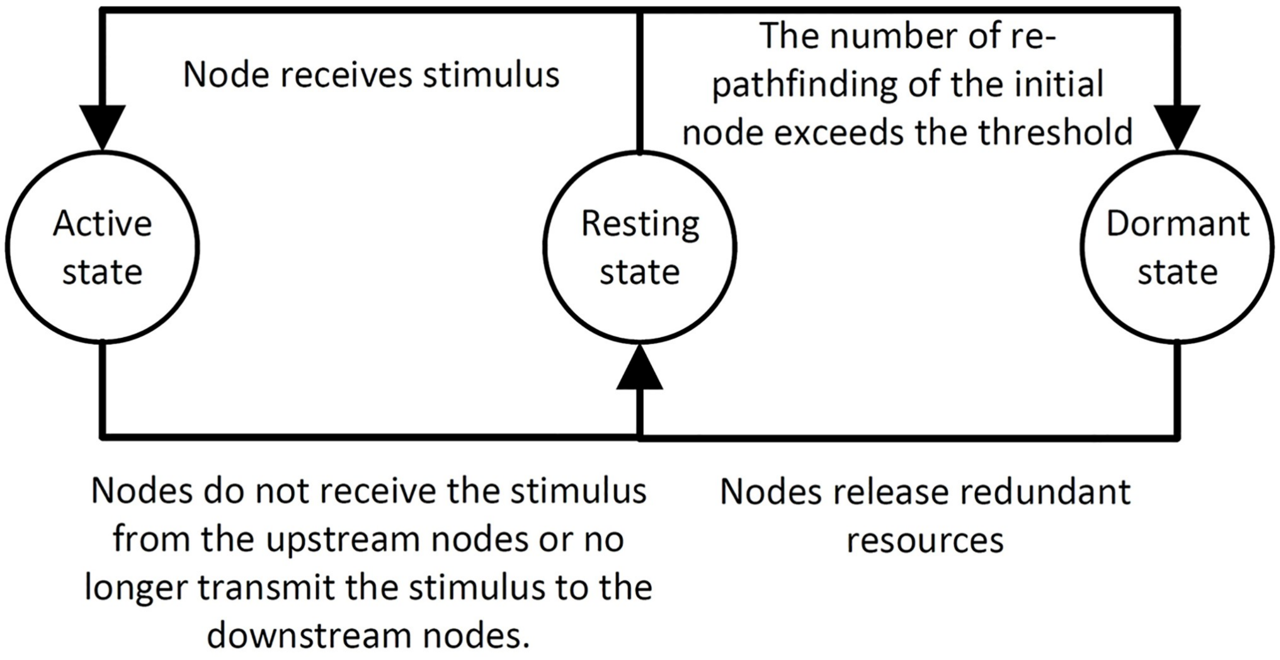

Node receives stimulus
The number of re-pathfinding of the initial node exceeds the threshold
Active state
Resting state
Dormant state
Nodes do not receive the stimulus from the upstream nodes or no longer transmit the stimulus to the downstream nodes.
Nodes release redundant resources

Fig 5. Node state transition graph.

https://doi.org/10.1371/journal.pcsy.0000019.g005

The second and third cases can be solved by introducing a re-pathfinding rule. The re-pathfinding rule allows the initial nodes to re-propagate the stimulus to find a path connecting other initial nodes when it does not successfully activate any downstream node. For the fourth case, the active node can release some occupied resources according to its situation. The resource release algorithm is introduced here to solve this problem. When the number of failed re-pathfindings of an initial node reaches a certain threshold, it will enter a dormant state, indicating that it is currently unable to communicate with other nodes. The node in the dormant state suspends pathfinding until the subgraph stabilizes. Active nodes will release some nodes after the subgraph is stabilized to make resources available to dormant nodes. For example, if an active node has three active fan-out nodes, it can actively release the occupation of two of them. After releasing the redundant resources, the nodes in the dormant state will resume pathfinding until the subgraph stabilizes again. If, after releasing the resources, the dormant node is still unable to establish path connections to other active nodes, the network connectivity is considered poor, or there is a conflict between the current sample and the samples already stored in the network. In this case, the subgraph can still be formed. However, there will be some isolated nodes that cannot establish connections with other nodes, leading to a decrease in the subgraph's anti-interference ability and fault tolerance. The pseudo-code of the resource release algorithm is given by Algorithm 3. Fig 5 shows the transition relationship between the three node states.

Algorithm 3: *ResourcesReleasing*()

```
Data: dormantCnt: The number of nodes in dormant state. isDormant[$v_a$]:
      Whether $v_a$ is in dormant state. repathCnt[$v_a$]: Indicates the num-
      ber of re-pathfinding.
Output: True: The resource is released successfully and needs to con-
        tinue to iterate. False: No node resources can be released,
        the iteration is complete.
```

```
if dormantCnt > 0 then
  isReleased ← False
  /* The resources will be released if the subgraph is stable and
     several nodes are dormant.                                  */
  for $v_a$ ← 0 to n do
    if out degree of $v_a$ > 1 then
      /* Release redundant resources                             */
      isReleased ← releaseResources($v_a$);
    end
    if isDormant[$v_a$] == True then
      isDormant[$v_a$] ← False;
      dormantCnt ← dormantCnt - 1;
      repathCnt[$v_a$] ← 0;
    end
  end
  return isReleased;
end
return False;
```

2.2.5 Sample storage and retrieval. The sample storage process consists of two stages: (1) Stimulus propagation stage: the initial nodes propagate stimulus to other nodes along the directed edges until a stable subgraph is formed. (2) Subgraph consolidation stage: All nodes in the subgraph update their internal index tables, recording the activation traces. The corresponding pseudo-code is given by Algorithm 4.

Algorithm 4: *SubgraphGeneratingAndSaving*(*initialNodes*)

```
Data: isActive[$v_a$]: whether $v_a$ is activated. isInitialNode[$v_a$]:
      Whether $v_a$ is a initial node. changed: Whether the network has
      iterated to a stable state.
Input: initialNodes
dormantCnt ← 0;
for $v_a$ in initialNodes do
  isActive[$v_a$] ← True;
  isInitialNode[$v_a$] ← True;
end
while True do
  changed ← False;
  for $v_a$ ← 0 to n do
    /* $v_a$ receives stimulus from upstream nodes               */
    receiveStimulus($v_a$);
  end
  for $v_a$ ← 0 to n do
    if isDormant[$v_a$] == True then
      continue;
    end
    /* $v_a$ delivers stimulus to downstream nodes               */
    NodeStimulusSpreading($v_a$);
  end
  /* The network is not stable, continue to iterate              */
  if changed == True then
    continue;
  end
  /* The network is stable and releases resources                */
  if ResourcesReleasing() == False then
    break;
  end
end
ActivationTracesUpdating();
```

Stimulus propagation stage: Fig 6 demonstrates a complete process of generating a stable subgraph through continuous iteration of the initial nodes. At the time t_0, the initial nodes are activated, and downstream nodes are chosen for stimulus propagation according to the weighted random selection algorithm. The subsequent t_1 and t_2 moments represent the continuous stimulus propagation in the network. At the time t_3, since downstream nodes v_b and v_c of node v_h have been occupied by other nodes, node v_h cannot perform stimulus transfer. Therefore, according to the node resource-grabbing rules, node v_h transitions from the active state to the resting state. At the time t_4, downstream node v_h, excited by node v_j, reverts to a resting state. At this point, node v_j does not activate any nodes, and due to the avalanche effect, its state also becomes a resting state. After the end of time t_4, node v_d will no longer propagate stimulus. However, as the initial node, it will follow the re-pathfinding rules, searching for a new path and attempting to participate in the subgraph formation. When re-pathfinding reaches a certain number of attempts, the node will enter a dormant state. Here, it is assumed that node v_d has entered a dormant state and will halt pathfinding until the subgraph is stable. It can be observed that at time t_4, the subgraph is already stable since there will be no change in node states. At this point, it is necessary for other active nodes in the network to release redundant resources, providing node v_d the opportunity to re-engage in the subgraph formation. At the time t_5, node v_a, which originally occupied both node v_e and node v_g resources, can choose to release the occupation of either of the two nodes. Assuming that node v_e is released, node v_e will become resting, and the stimulus from node v_e to node v_b will also vanish. However, because node v_b is an initial node, its state will not change. After the resource is released, node v_d resumes pathfinding and node v_j is activated at time t_6. At the time t_7, node v_h is activated by node v_j, and the stimulus is passed to the initial node v_b, forming a path. At this moment, the connected subgraph between active nodes becomes stable, no dormant nodes are present in the network, and the stimulus propagation stage concludes.

Subgraph Consolidation Stage: The primary task of this stage is to store a stable subgraph structure in the network. When the subgraph achieves a stable state, each active node will have corresponding active fan-in and active fan-out nodes, which are activation traces. Storing the subgraph is completed by updating the activation trace of each node in the node's internal index table. The pseudo-code of the index table update algorithm is provided by Algorithm 2, and the pseudo-code of subgraph generation and preservation is given by Algorithm 4.

The process of sample retrieval closely resembles that of sample storage. However, the sample retrieval process is simpler, only including the stimulus propagation stage. During the stimulus propagation stage, it will not be activated when a node receives a stimulus transfer from an upstream node and cannot find a similar entry in its internal index table. The initial nodes will not enter the dormant state, and no nodes will release excessively occupied resources. In summary, the sample retrieval algorithm will not cause any changes to the existing network. However, it will only perform stimulus propagation based on the activation traces stored in the internal index table of the node. Since the sample retrieval algorithm process is very similar to the storage algorithm, only the specified part mentioned above needs to be omitted. Therefore, no separate pseudo-code is provided here.

3 Results

The experiment is primarily divided into four aspects:

1. Capacity testing: This aims to investigate the number of samples the network can stably store and the factors influencing network capacity.

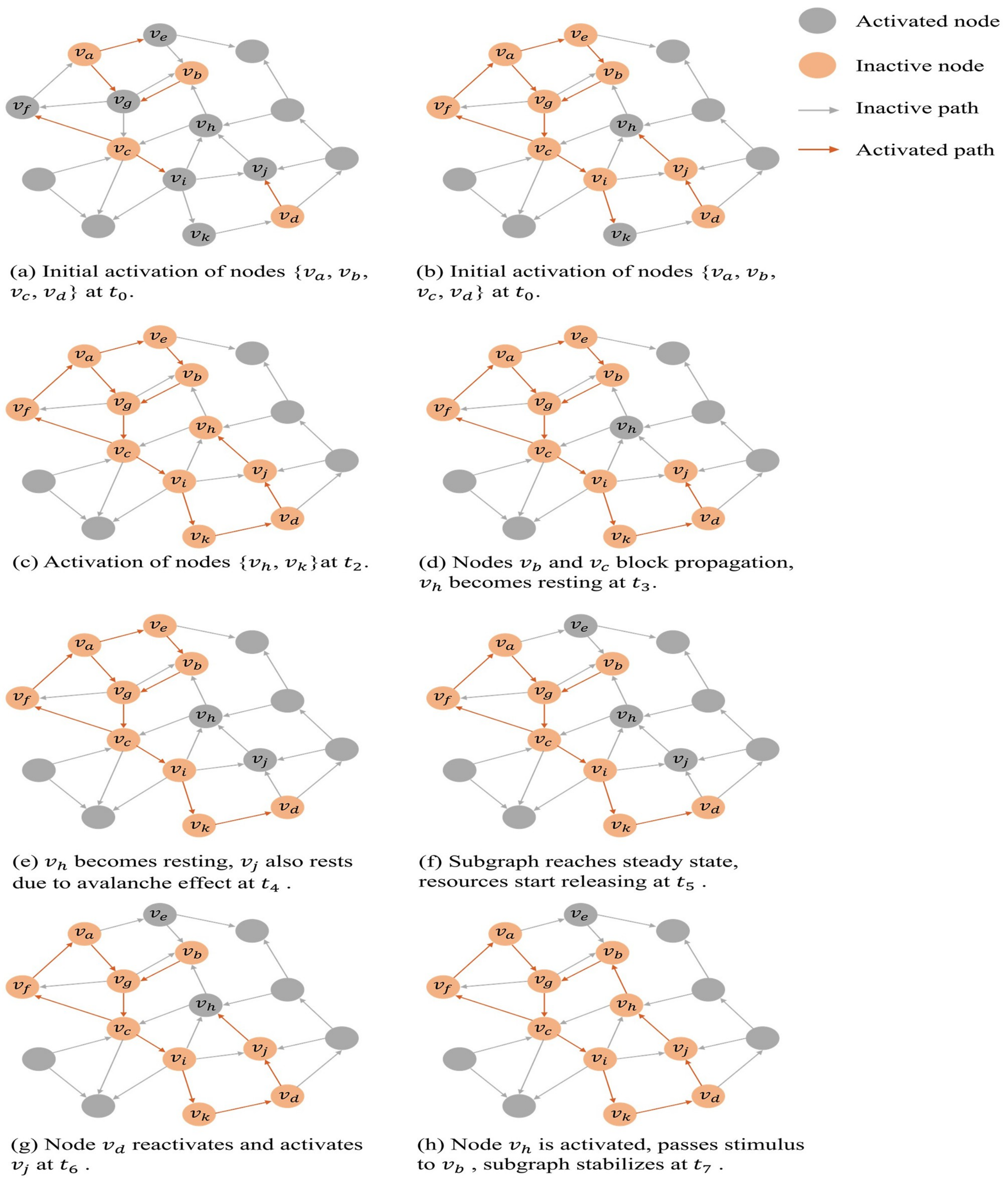

Activated node
Inactive node
Inactive path
Activated path
(a) Initial activation of nodes {v_a, v_b, v_c, v_d} at t_0.
(b) Initial activation of nodes {v_a, v_b, v_c, v_d} at t_0.
(c) Activation of nodes {v_h, v_k}at t_2.
(d) Nodes v_b and v_c block propagation, v_h becomes resting at t_3.
(e) v_h becomes resting, v_j also rests due to avalanche effect at t_4 .
(f) Subgraph reaches steady state, resources start releasing at t_5 .
(g) Node v_d reactivates and activates v_j at t_6 .
(h) Node v_h is activated, passes stimulus to v_b , subgraph stabilizes at t_7 .

Fig 6. Example of a dynamic process for directed graph stimulus propagation. (a) t_0: Initial nodes {v_a,v_b,v_c,v_d} are activated, which performs stimulus propagation according to a weighted random selection algorithm. (b) t_1: {v_e,v_f,v_g,v_i,v_j} becomes active after receiving stimulus from the initial nodes. (c) t_2: {v_h,v_k} becomes active after receiving stimulus from active nodes. (d) t_3: The downstream nodes {v_b,v_c} of v_h are all occupied, so stimulus cannot be propagated. v_h becomes resting again. (e) t_4: After v_h becomes resting state, according to the avalanche effect, v_j also becomes resting state. (f) t_5: The subgraph iterates to a steady state. Start to release resources, and node v_a releases the occupancy of v_e. (g) t_6: After the resources are released, the dormant node v_d restarts pathfinding and successfully activates v_j. (h) t_7: Node v_h is successfully activated after receiving the stimulus from v_j and passing the stimulus to v_b. The subgraph is iterated to a stable state.

https://doi.org/10.1371/journal.pcsy.0000019.g006

2. Fault tolerance testing: This mainly explores the effect of sample retrieval when the input is incomplete or has noise.
3. Robustness testing: This mainly explores the effect of sample retrieval when some nodes or edges are damaged.
4. Performance testing on different classical network structures: This mainly explores the algorithm's performance on various classic network structures.

The network used in the experiment is ER random graphs [29]. This classic random network model, proposed by Paul Erdős and Alfréd Rényi in 1959, is defined by having a probability p of connection between any two nodes in the network. Extending this definition to directed graphs, two distinct directional edges can be between any two nodes. The probabilities of these two edges existing are independent of each other, and both are equal to p.

3.1 Capacity testing

Let $G^k = (V^k, E^k)$ represent the subgraph generated of the kth sample, where V^k denotes the set of nodes, and E^k denotes the set of edges. Let $G^k_{sg} = (V^k_{sg}, E^k_{sg})$ represent the subgraph generated during the retrieval of the kth sample. Define the accuracy $P^k = \frac{|E^k \cap E^k_{sg}|}{|E^k_{sg}|}$. Define the completeness $C^k = \frac{|E^k \cap E^k_{sg}|}{|E^k|}$. Define an isolated node as an initial node in the subgraph with both in-degree and out-degree equal to zero. The sample representation quality, Q, is defined as the percentage of non-isolated nodes relative to the initial nodes. If the number of initial nodes in the sample is s, and the number of isolated nodes in the subgraph generated by the sample is l, then $Q = \frac{s-l}{s}$. In capacity testing, each stored sample should satisfy a high sample representation quality and high completeness and accuracy during sample retrieval to be considered successfully stored by the network. Based on this, we can define the reliable capacity of the network. Let the reliable capacity, T, represent the maximum number of samples that the network can successfully store, and for each stored sample, it satisfies $Q_k > 0.9$, $i \in [1, T]$, $\bar{P} = \frac{\sum_{k=1}^{T} P^k}{T} > 0.9, \bar{C} = \frac{\sum_{k=1}^{T} C^k}{T} > 0.9.$

Assume the network has n nodes, each node has an internal index table with a capacity of K, and each subgraph contains, on average, s initial nodes and c communication nodes. For every subgraph stored in the network, there will be an increase or modification in the internal index table entries of the activated nodes. Considering this as a resource, the total number of resources in the network is nK, and each subgraph occupies $s + c$ resources. Without considering resource reuse or optimization measures, the network will consume $s + c$ resources for every stored sample, so the network capacity can be roughly represented as $\frac{nK}{s+c}$. If resource reuse is allowed, the calculation of network capacity becomes more complex. In an extreme case, where the resources occupied by the current subgraph are all reused, the upper bound of the network capacity can be roughly expressed as the combination number $\binom{nK}{s+c}$. The network capacity obtained from these two different calculation methods differs greatly. In actual testing, there are many other influencing factors, such as different network connectivity and conflicts between samples. Therefore, a theoretical network capacity analysis is difficult, and a specific analysis should be conducted in conjunction with actual testing situations.

Table 1 shows the performance of sample retrieval after storing 1,000 samples in networks. The node index table size is set to $K = 20$. The scale of a single sample refers to the size of the initial node set. As shown in Table 1, the capacity of sparse graphs is typically larger than that of dense graphs with the same node size. The primary distinction between sparse and dense

Table 1. Different-scale network retrieval performance after storing 1000 samples.

Number of network nodes	Number of edges	Single sample scale	Subgraph average number of nodes	Subgraph average number of edges	Subgraph average number of WCCs	Average accuracy	Average completeness	Accuracy Std Dev	Completeness Std Dev
500	3101 (Sparse)	15	24.024	15.651	3.372	99.6%	98.4%	0.0013	0.0028
500	3101 (Sparse)	60	85.343	70.763	10.312	97.5%	94.8%	0.0036	0.0044
500	12606 (Dense)	15	28.407	27.774	2.760	98.1%	67.2%	0.0032	0.0049
500	12606 (Dense)	60	66.590	88.406	1.720	99.0%	63.2%	0.0011	0.0087
2000	15037 (Sparse)	15	30.428	20.664	3.391	99.5%	98.7%	0.0023	0.0026
2000	15037 (Sparse)	60	101.564	71.571	14.048	99.4%	98.4%	0.0013	0.0010
2000	199452 (Dense)	15	38.593	41.987	1.706	100%	82.5%	0.0001	0.0095
2000	199452 (Dense)	60	68.729	104.784	1.356	100%	57.9%	0.0000	0.0021

https://doi.org/10.1371/journal.pcsy.0000019.t001

graphs lies in the number of directed edges within the network, which directly influences network connectivity. This can also be observed from the average number of weakly connected components in the subgraphs presented in Table 1. For networks with the same node size, the more edges they have, the fewer weakly connected components their subgraphs have on average. Generally, the subgraphs generated by samples are not necessarily connected but are composed of multiple connected components. Weakly connected components (WCCs) are defined as components where undirected edges replace all directed edges, and any two points within the component are reachable from one another. The number of connected components reflects the aggregation of the subgraph. A greater number of connected components indicates a more dispersed subgraph, while a smaller number of connected components signifies a more clustered subgraph.

The connectivity or structure of subgraphs is undeniably a crucial factor influencing network capacity, as it determines the resource usage of each subgraph. There are two main factors that impact subgraph connectivity: the scale of a single sample and network connectivity. Table 1 demonstrates that a larger scale of a single sample and better network connectivity will reduce network capacity. This observation is intuitive for the former but counterintuitive for the latter. However, when the scale of a single sample node is 0 or network connectivity is extremely poor, the network capacity tends to be 0. This suggests that the relationship between network capacity and subgraph connectivity is not linear.

Fig 7 shows the changes in network capacity and subgraph structure as the number of edges in a network increases. The node size of the network is 500, and the single sample scale is 60. It can be observed that the network capacity first rises and then declines, eventually stabilizing near the theoretical capacity value in the simple case, which is $\frac{nK}{s+c}$. During the stage of network capacity growth, both the average number of WCCs and the average number of nodes in the subgraph decrease, suggesting that the subgraph progressively transitions from "dispersed" to "clustered." Subsequently, there is a sharp decline in network capacity, and the average number of WCCs in the subgraph also drops dramatically. This indicates that the network connectivity has reached a critical point, with almost all nodes in the subgraph belonging to the same WCC. As a result, the "agglomeration effect" emerges. It means most initial nodes can connect

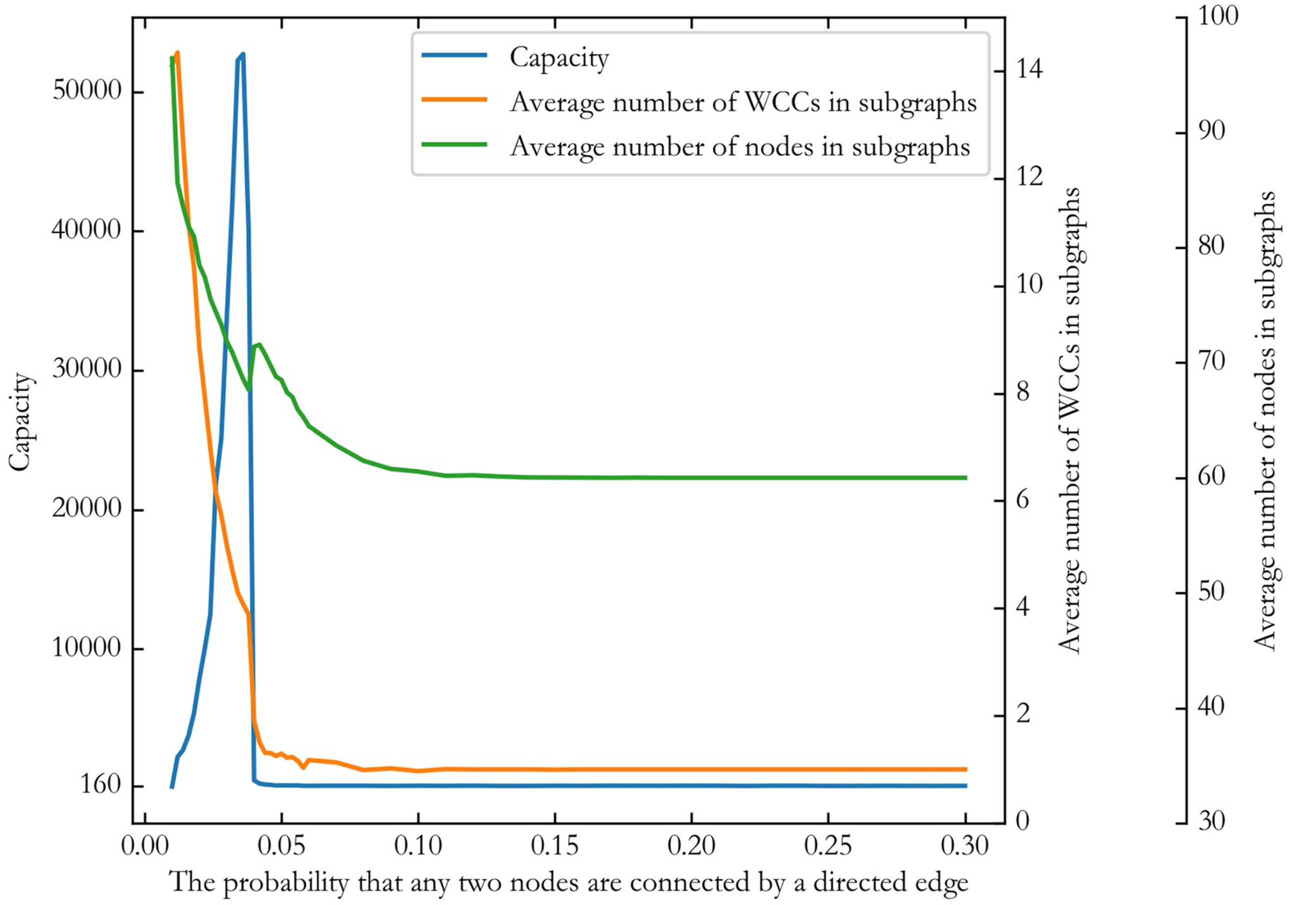


Fig 7. The relationship between the number of edges and network capacity and subgraph structure.

https://doi.org/10.1371/journal.pcsy.0000019.g007

directly without passing through other communicating nodes. When the network capacity reaches its peak, the average number of nodes in the subgraphs is close to the scale of a single sample, and the number of WCCs is slightly above 1.

Erdős and Rényi [30] demonstrated that when $p > \frac{(1+\epsilon)\ln n}{n}$, the ER random graph $G(n, p)$ is almost always connected. To ensure that the subgraphs generated by the samples have a high probability of only 1 WCC, it needs to guarantee that $p > \frac{(1+\epsilon)\ln s}{s}$, where s represents the scale of a single sample, which is 60. Let's take $p = \frac{\ln s}{s} \approx 0.07$. The generated subgraph in this scenario is shown in Fig 8A. If we take $p = 0.04$, corresponding to the p when the network capacity reaches its peak, the generated subgraph is shown in Fig 8B. It can be found that the essence of large capacity is actually the permutation and combination of multiple WCCs. When p is slightly less than $\frac{\ln s}{s}$, the subgraphs generated by the samples are composed of a small number of WCCs. Assuming the subgraph is evenly divided into t WCCs, the size of each WCC is $\frac{s+c}{t}$. The network capacity can be perceived as selecting t WCCs from all possible ones. This is essentially a Uniform disordered grouping problem. The calculation result is shown in formula 1. Although the actual capacity is significantly smaller than this value, it

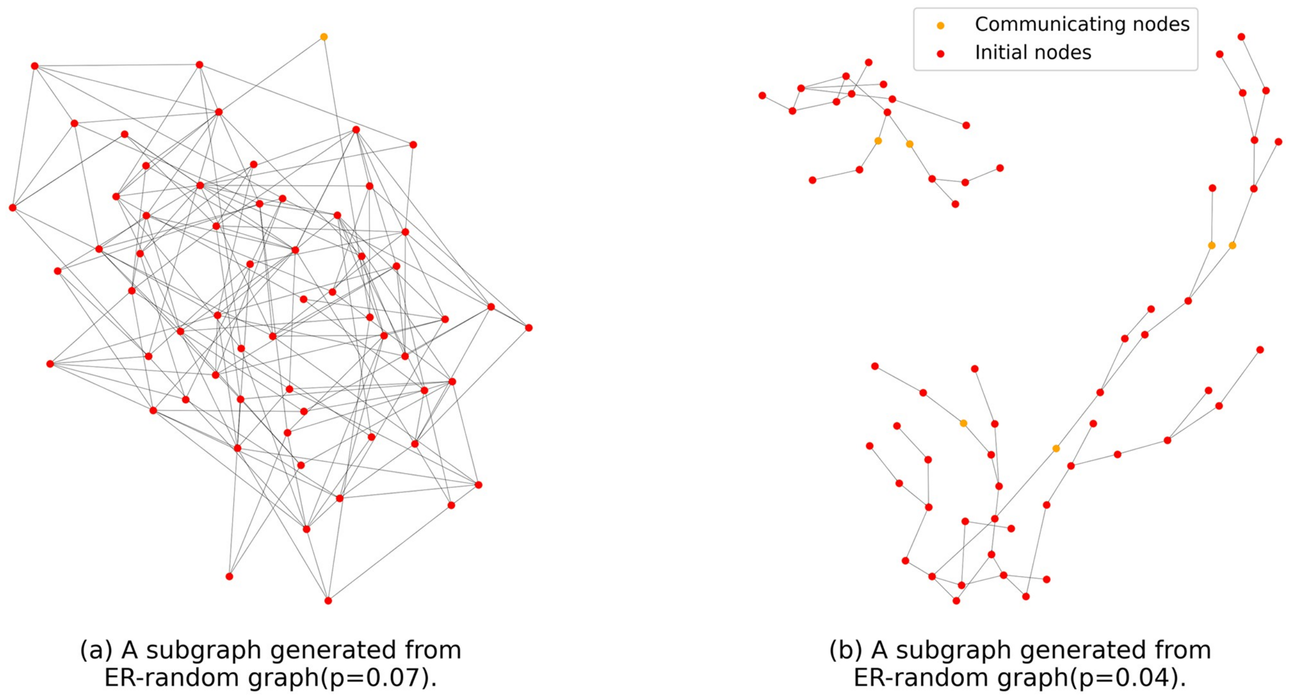

Communicating nodes
Initial nodes
(a) A subgraph generated from ER-random graph(p=0.07).
(b) A subgraph generated from ER-random graph(p=0.04).

Fig 8. Storage examples under different network connectivity. (a) The sample-generated subgraph in an ER random graph with p = 0.07. (b) The sample-generated subgraph in an ER random graph with p = 0.04.

https://doi.org/10.1371/journal.pcsy.0000019.g008

still demonstrates the huge storage potential of the network.

$$\begin{aligned} T &= \frac{\binom{n}{\frac{s+c}{t}}\binom{n-\frac{s+c}{t}}{\frac{s+c}{t}}\cdots\binom{n-(t-1)\frac{s+c}{t}}{\frac{s+c}{t}}}{t!} \\ &= \frac{n!}{(t!)^{\frac{s+c+t}{t}}(n-s-c)!} \end{aligned} \tag{1}$$

Fig 9 demonstrates the average accuracy, accuracy standard deviation, average completeness, and completeness standard deviation of sparse and dense graphs with the same number of nodes. These data provide a clear visual comparison of the memory performance differences between different graph structures. It can be observed that in the sparse graph, the average completeness of sample retrieval drops below 80% when the number of stored samples exceeds 8000. In contrast, for the dense graph, the average completeness of sample retrieval declines below 80% when the number of stored samples approaches 300. This capacity difference between the two networks further confirms that the arrangement and combination of WCCs are the essences of large capacity. Although the dense graph has more connections, its displayed capacity is not directly proportional to the number of resources owned by the network. Conversely, the sparse graph has only a small number of connections, but the network capacity achieved by the arrangement and combination of multiple connected subgraphs is several times that of the dense graph. This indicates that a sparse connection is a more reasonable mode, which can effectively save resources and obtain a larger network capacity. Moreover, the biological neuron network of the human brain also follows a sparse connection mode,

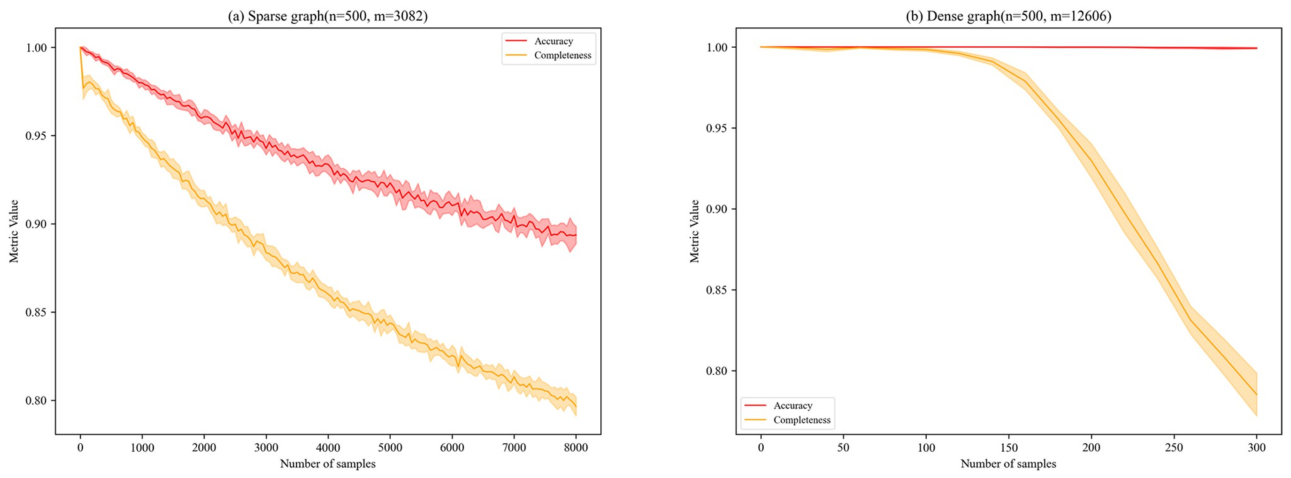


Fig 9. Comparison of capacity between sparse and dense graphs. (a) Results in a sparse graph. (b) Results in a dense graph.

https://doi.org/10.1371/journal.pcsy.0000019.g009

implying that sparse connections are efficient and maximize the use of resources. Sparse connectivity significantly reduces wiring costs. In the brain, each neuron is connected to only a limited number of other neurons, greatly reducing the number and length of nerve fibers. This reduction in nerve fibers not only saves biological materials but also decreases the energy consumption required for the brain to grow and maintain these connections. At the same time, sparse connectivity enables more efficient information transmission. In a sparsely connected network, signals are quickly transmitted to target neurons through specific pathways, minimizing unnecessary transmission delays. This efficient mode of information transmission allows the brain to respond rapidly to external stimuli and perform excellently in complex cognitive tasks.

3.2 Fault tolerance testing

Fault tolerance testing primarily explores the effect of sample retrieval when the input is incomplete or noisy. The network used in the experiment is an ER random graph with 500 nodes and either 3082 (sparse graph) or 12606 (dense graph) edges. The experiment first stores 1000 samples in the network and then modifies the sample inputs during the retrieval process. Finally, compare how the average accuracy and completeness change when the sample input is incomplete or has noise. There are three categories of modifications to inputs:

1. Removing a part of the original sample input to explore the impact of incomplete inputs on sample retrieval performance.
2. Adding extra noisy nodes to the original sample to investigate the impact of noise on retrieval.
3. Removing part of the original sample input and replacing it with an equal number of noisy nodes to examine the impact of sample retrieval in this mixed scenario.

Fig 10 shows the average accuracy, accuracy standard deviation, average completeness, and completeness standard deviation of sparse and dense graphs under conditions of incomplete sample input. As the level of missing sample input increases, both the retrieval accuracy and completeness of sparse and dense graphs decrease to varying degrees. The change trends of the two networks are generally similar, and the decline in accuracy is relatively gentle. When the

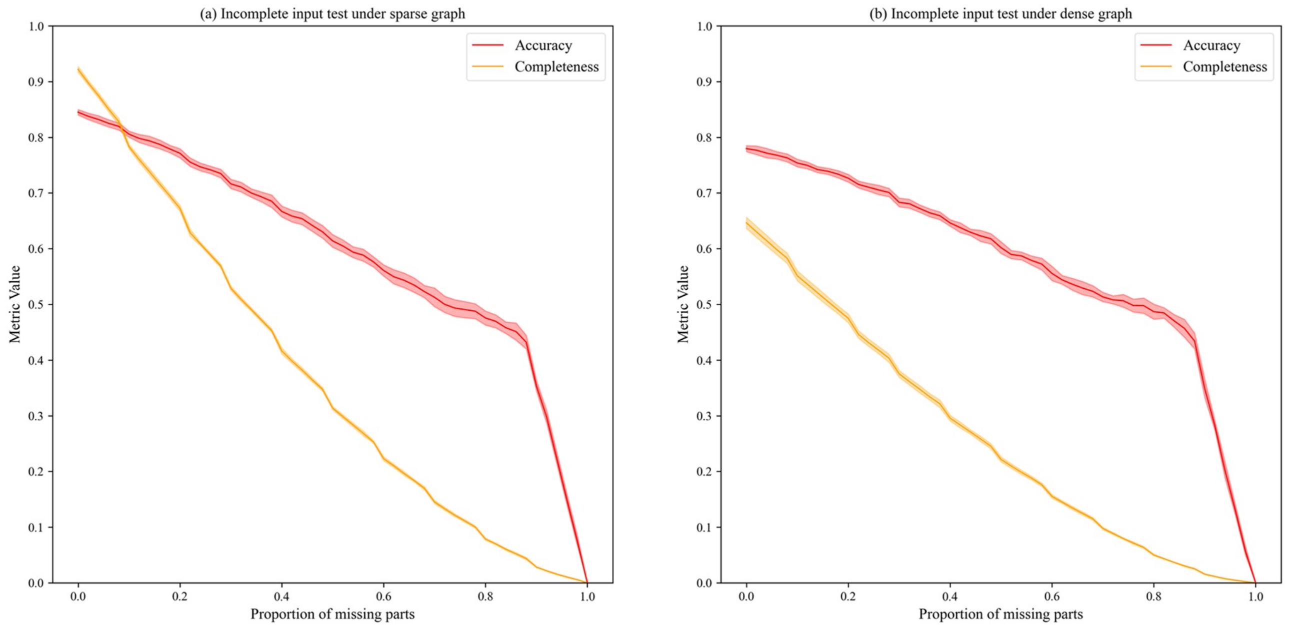


Fig 10. The impact of incomplete sample input on sample retrieval. (a) Test results in a sparse graph. (b) Test results in a dense graph.

https://doi.org/10.1371/journal.pcsy.0000019.g010

proportion of missing parts reaches 80% of the input, the rate of accuracy decline increases significantly. Compared to Fig 10B and 10A has higher completeness and accuracy when the proportion of missing inputs is between 0.0 and 0.1. This is because the network capacity of the dense graph is small, making it difficult to achieve high reading accuracy and completeness after storing 1000 samples. The overall decline rate of completeness is greater than that of accuracy, indicating that the erroneous content obtained during retrieval does not increase as the proportion of missing inputs increases. This suggests that the algorithm is relatively reliable when facing missing sample inputs, although the rapid decline in completeness represents a significant amount of correct content that cannot be read. However, even when the proportion of missing inputs is as high as 80%, the retrieval accuracy can still be maintained at around 40% to 50%, which means that even if there are numerous missing inputs, almost half of the read content is correct and reliable.

Fig 11 shows the average accuracy, accuracy standard deviation, average completeness, and completeness standard deviation of sparse and dense graphs with the addition of noise nodes. It can be observed that these additional noise nodes have relatively little effect on the accuracy and completeness of sample retrieval. The decline rate of completeness is lower than that of accuracy because adding noise nodes generally do not directly disrupt the original subgraph structure but makes the final subgraph larger. This demonstrates that the network has strong resistance to noise and is sensitive to the absence of sample inputs.

Fig 12 presents the performance of both incomplete and contained noise nodes in the sample input under sparse and dense graphs, respectively. It can be seen that the decline rate of accuracy and completeness, in this case, is the fastest, indicating that the impact of missing inputs and the effect of noise nodes can be superimposed.

3.3 Robustness testing

It is known that neurons in the biological brain may experience various functional failures. How does this affect memory? This section primarily examines how the performance of

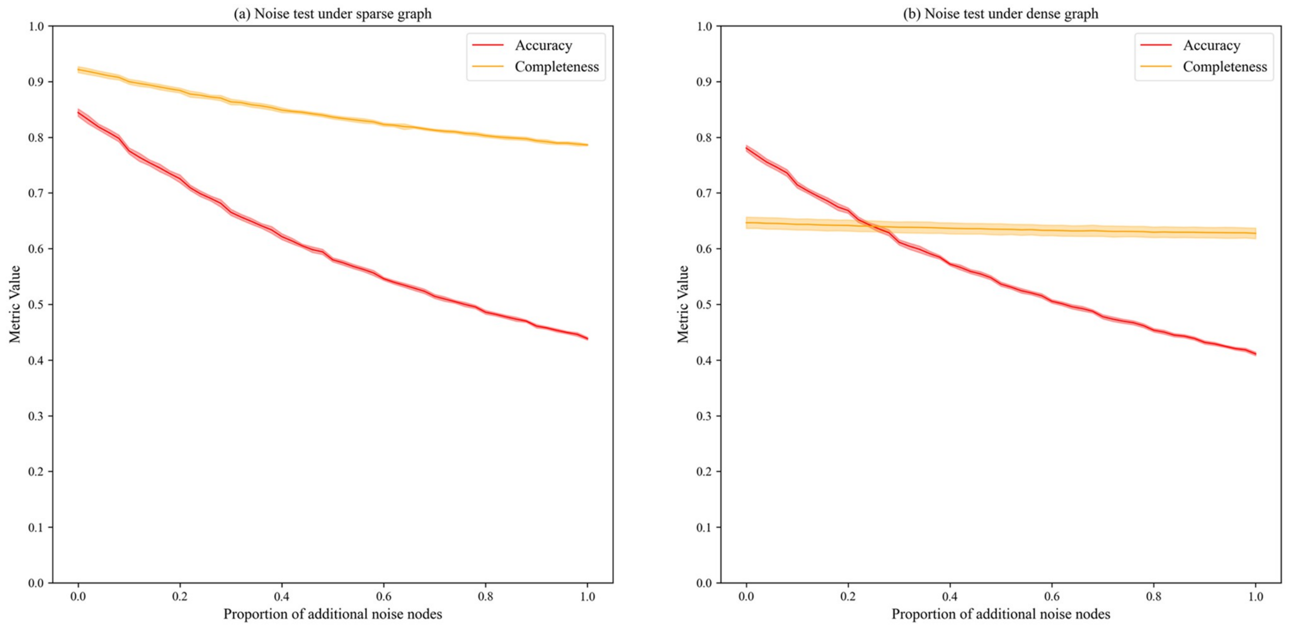


Fig 11. The impact of noisy sample input on sample retrieval. (a) Test results in a sparse graph. (b) Test results in a dense graph.

https://doi.org/10.1371/journal.pcsy.0000019.g011

sample retrieval changes when the network is damaged to different degrees. The test includes two main aspects: damage to some nodes and damage to some directed edges. The experiment first stores 1000 samples in the network, then deletes a certain proportion of nodes or directed edges and subsequently attempts to retrieve these samples while recording average accuracy

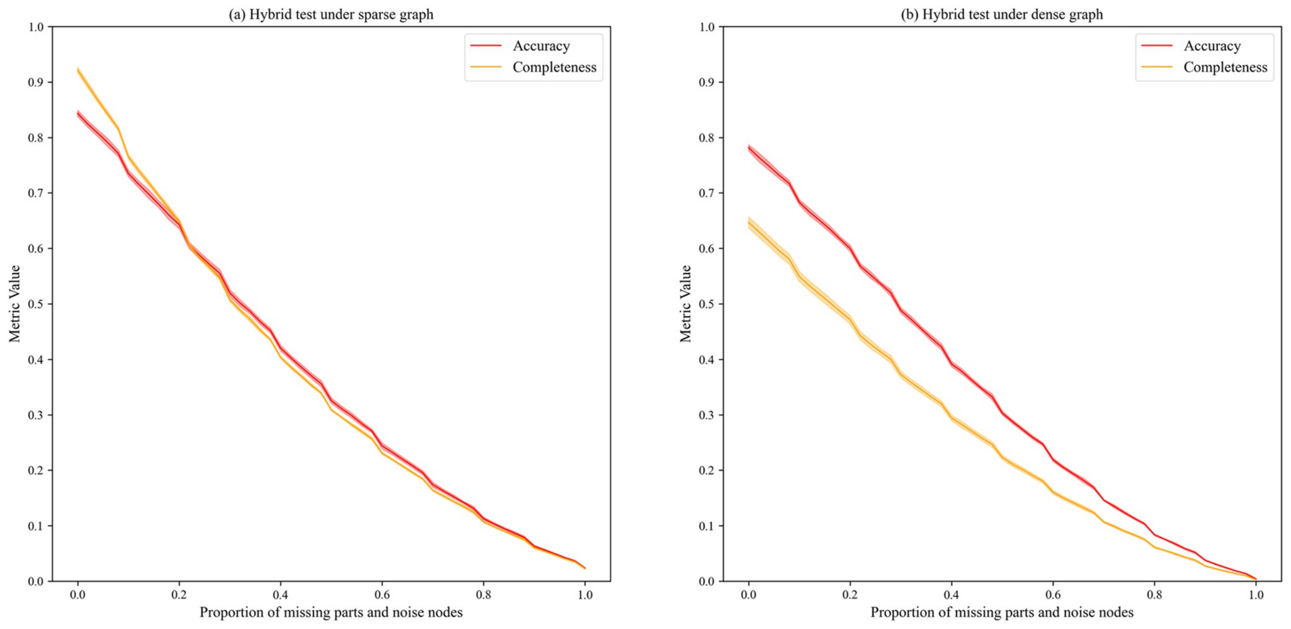


Fig 12. The impact of both incomplete and contained noise nodes in the sample input on sample retrieval. (a) Test results in a sparse graph. (b) Test results in a dense graph.

https://doi.org/10.1371/journal.pcsy.0000019.g012

Table 2. Restoration schemes for the index table after damage.

Restoration schemes	The modified content of the index table
Maintain the original trace	$\{v_a,v_b,v_c\} \rightarrow \{v_x,v_y,v_z\}$, $\{v_b,v_c,v_d\} \rightarrow \{v_u,v_x,v_z\}$
Take the union of the outputs	$\{v_b,v_c\} \rightarrow \{v_x,v_y,v_u\}$
Take the intersection of the outputs	$\{v_b,v_c\} \rightarrow \{v_x\}$
Take the output with the highest occurrence frequency	$\{v_b,v_c\} \rightarrow \{v_x,v_y\}$ or $\{v_u,v_x\}$

https://doi.org/10.1371/journal.pcsy.0000019.t002

and completeness changes. The network used in the experiment is an ER random graph with 500 nodes and 3101 edges (sparse graph) or 12606 edges (dense graph).

After a node or a directed edge is deleted, the activation traces recorded in the node's internal index table will be affected. Assume that an index table contains two items: $v_a,v_b,v_c \rightarrow v_x,v_y,v_z$ and $v_b,v_c,v_d \rightarrow v_u,v_x,v_z$. If nodes v_a, v_d, and v_z are deleted, does it need to delete the corresponding node in the activation trace recorded in its index table? If deleted, then these two items will become: $v_b,v_c \rightarrow v_x,v_y$ and $v_b,v_c \rightarrow v_u,v_x$. It can be observed that the input parts of these two items are the same, so addressing the different output parts is a challenge. Usually, during the initial period after network damage, nodes are hard to respond, and at this time, the original traces stored in the node index table will not change. As the damage duration increases, nodes may gradually make corresponding adaptive adjustments to the damaged network. Given the above two different situations, this paper proposes four restoration schemes, as shown in Table 2, and compares these four solutions.

Fig 13 demonstrates the impact of partial node damage on sample retrieval performance. Fig 13A and 13B respectively display the changes in average accuracy and completeness of sample retrieval in the sparse graph for the four restoration schemes. In terms of average accuracy, the scheme that maintains the original traces performs the best, the scheme that takes the union performs the worst, and the other two schemes exhibit similar performance. Conversely, in terms of average completeness, the results are reversed. The scheme that takes the union performs the best, while the one that maintains the original traces performs the worst. This is because the union-taking scheme increases the number of activated nodes, which includes both incorrect and correct nodes. The former leads to a decrease in accuracy, while the latter leads to an increase in completeness. Fig 13C and 13D present the results on the dense graph, revealing that when the network has a large number of edges, the differences between the four restoration schemes progressively diminish. This occurs because when network connectivity is high, the number of communication nodes in the subgraph generated by the sample is small, with most initial nodes being directly connected. Consequently, the accuracy remains consistently high. The frequency at which each node is shared by different samples is also reduced, so when a node is deleted, the number of samples it affects decreases, making the differences between the four solutions less noticeable.

Fig 14 shows the impact of partial directed edge damage on sample retrieval. Since a node only has a local view, it can receive and transmit the information of neighboring nodes solely through its fan-in and fan-out edges. Node damage can be understood as the interruption of all fan-in and fan-out connections, so the impact on neighboring nodes is essentially the same, whether node damage or directed edge damage. Consequently, the same restoration schemes can be used. Fig 14A and 14B respectively display the changes in average accuracy and completeness of sample retrieval for the four restoration schemes in the sparse graph. Their trends are almost consistent with Fig 14A and 14B. Regarding accuracy, the notable difference between the two is that in the interval [0.8,1.0], Fig 14A maintains relatively high accuracy.

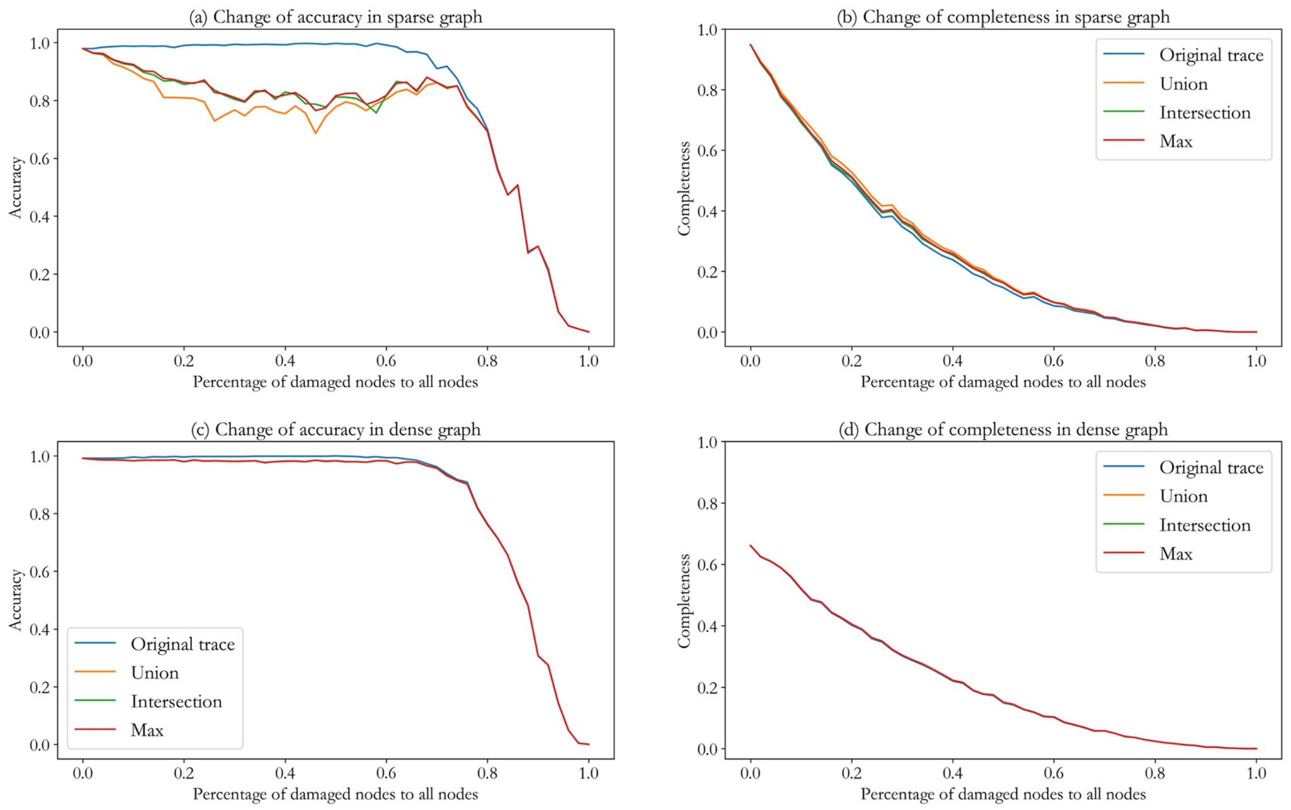


Fig 13. The impact of partial node damage on sample retrieval. (a) The change in accuracy in a sparse graph. (b) The change in completeness in a sparse graph. (c) The change in accuracy in a dense graph. (d) The change in completeness in a dense graph.

https://doi.org/10.1371/journal.pcsy.0000019.g013

Regarding completeness, the curve of Fig 14B is comparatively flat. Fig 14C and 14D are the results of dense graphs. The results are also consistent with those of Fig 13C and 13D. The difference is the same as that observed in the sparse graph, which indicates that the network is significantly more tolerant of directed edge damage than node damage, as nodes hold information while edges do not.

3.4 Performance testing on different classical network structures

The information storage and retrieval algorithm proposed in this paper is closely related to the network structure. Firstly, the algorithm utilizes the subgraph structure as the information storage carrier, and secondly, the subgraph formation depends on stimulus propagation. Both of these characteristics emphasize the importance of the network structure for the algorithm. Therefore, different network structure characteristics are key factors affecting the algorithm's performance.

Fig 15 showcases six classic network structures. Fig 15A is the ER graph with $p = 0.1$. Fig 15B represents a globally coupled network, also known as a fully connected network. Fig 15C shows the nearest-neighbor coupled network, characterized by N nodes arranged in a ring, with each node establishing connections to its left and right L neighbors, respectively. Fig 15D illustrates a star coupled network, featuring a central node to which all other nodes are connected. This characteristic causes any path between two points in the network to include the central node, creating a bottleneck for the entire network capacity. Fig 15E presents a one-

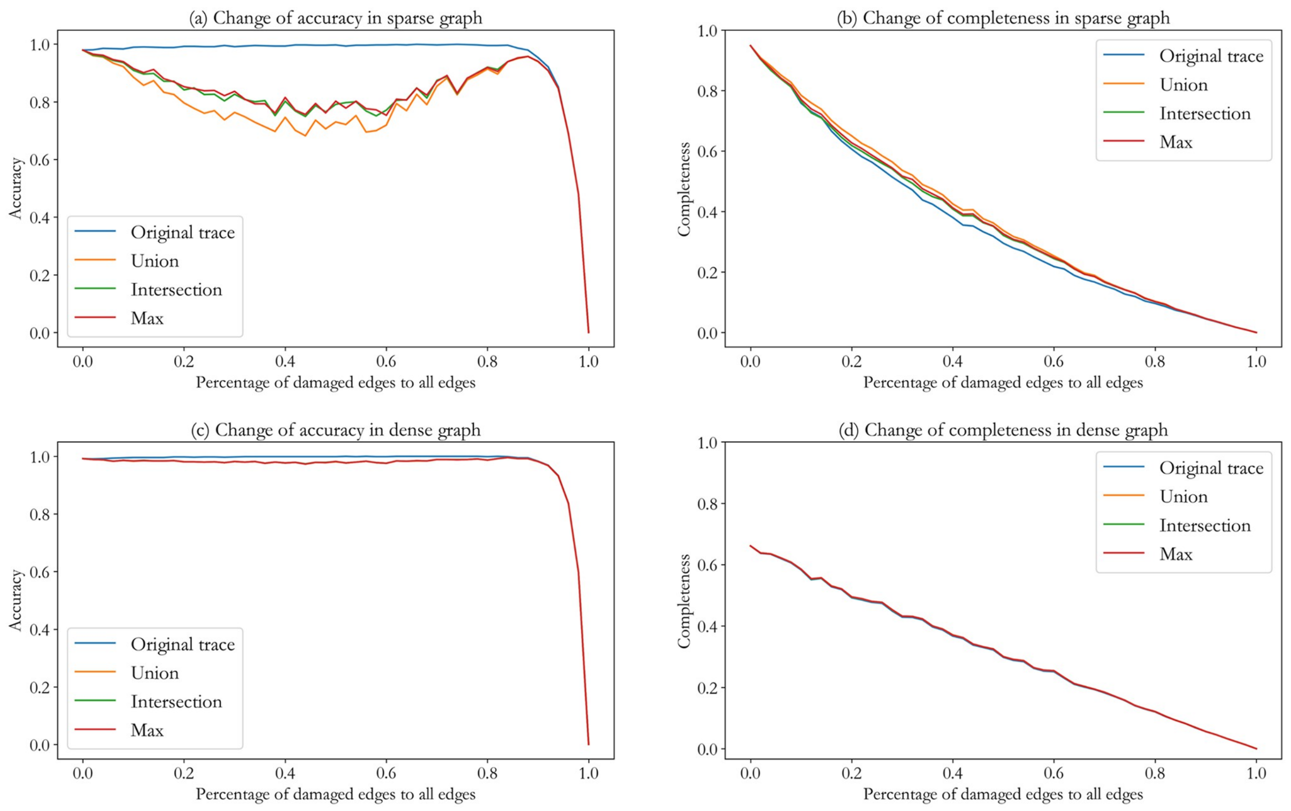


Fig 14. The impact of partial directed edge damage on sample retrieval. (a) The change in accuracy in a sparse graph. (b) The change in completeness in a sparse graph. (c) The change in accuracy in a dense graph. (d) The change in completeness in a dense graph.

https://doi.org/10.1371/journal.pcsy.0000019.g014

dimensional Kleinberg network [31], a small-world network [32]. The network is constructed by adding a few random edges to the nearest-neighbor coupled network. Fig 15F displays the Price network [33], which is a scale-free network. The network's generation relies on the preferential attachment mechanism, where newly added nodes are more likely to connect to nodes with higher degrees. Since each newly added directed edge point from the new node to the old node, there are no loops in the network, leading to a significant decrease in network connectivity and capacity.

Evaluation parameters for different network structures typically include average path length and clustering coefficient.

Average Path Length: Defined as the average of the shortest path lengths between any two points in the network. The default shortest path length is usually positive infinity if the two nodes are disconnected. This special case is common in directed graphs. To prevent the calculation result from being positive infinity, this paper uses the harmonic mean [34] of the distance between any two nodes in the network to represent the average path length.

N represents the number of network nodes, and $d(v_i, v_j)$ represents the shortest distance between node v_i and node v_j. GE represents network communication efficiency, with its essential idea being that the closer the node path distance in the network, the higher the communication efficiency. The average path length calculated by the formula 2 [34] solves the problem of the value being positive infinity when the network is disconnected. Therefore, it is more

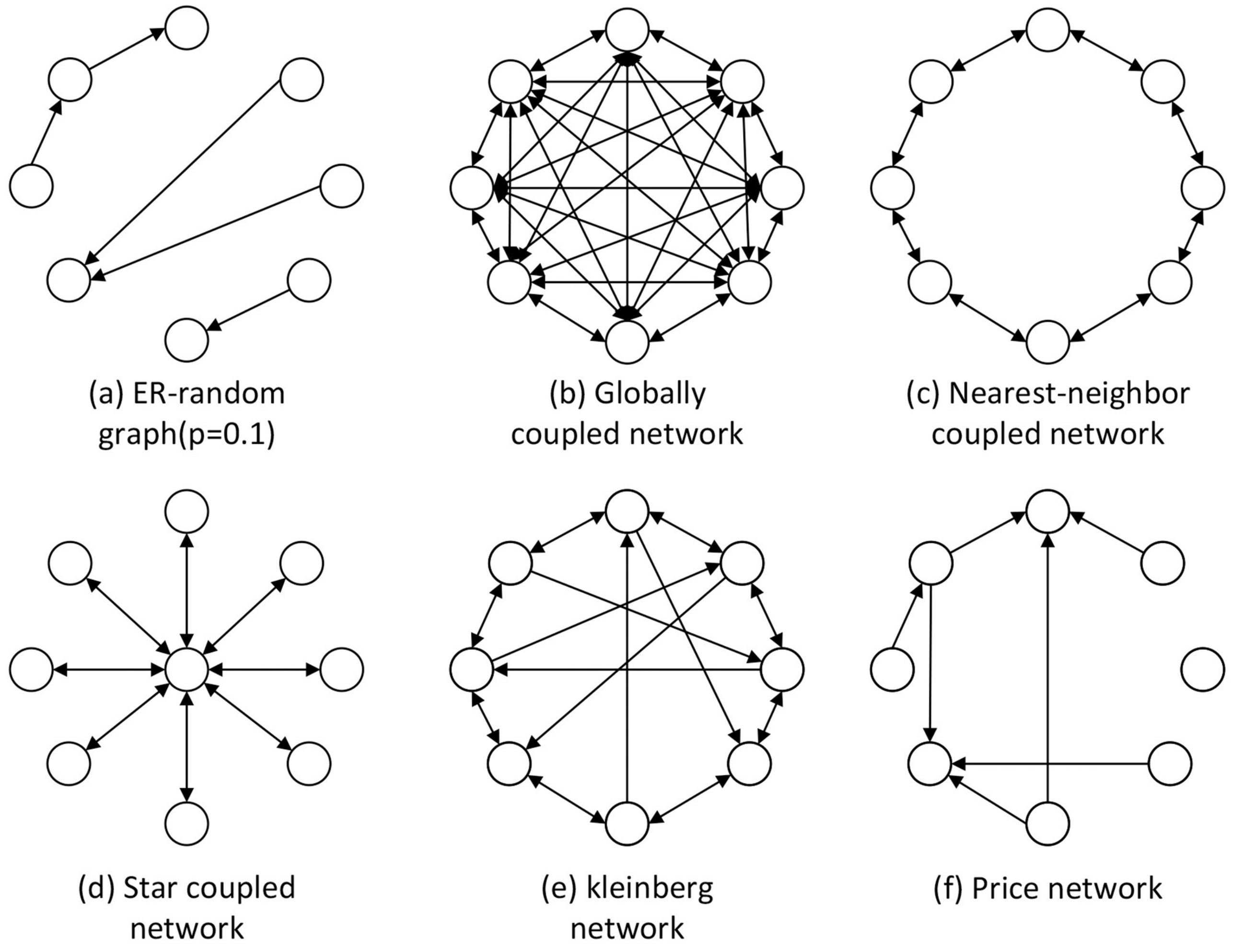

(a) ER-random graph(p=0.1)
(b) Globally coupled network
(c) Nearest-neighbor coupled network
(d) Star coupled network
(e) kleinberg network
(f) Price network

Fig 15. Six classic network structures. (a) ER random graph with p = 0.1. (b) Globally coupled network. (c) Nearest-neighbor coupled network with L = 1. (d) Star coupled network. (e) Kleinberg network. (f) Price network.

https://doi.org/10.1371/journal.pcsy.0000019.g015

suitable for evaluating directed graph network structure.

$$L = \frac{1}{GE}, GE = \frac{1}{N(N-1)} \sum_{v_i \geq v_j} \frac{1}{d(v_i, v_j)} \tag{2}$$

Clustering coefficient: This metric is used to measure whether the nodes in the network exhibit aggregation characteristics. This paper adopts the calculation method of the clustering coefficient in directed graphs proposed by Fagiolo [35].

Table 3 displays the test results of six network models with different structures but similar scales. The number of nodes in all test networks is 1000, and the single sample scale is 60. The ER random graph exhibits a small clustering coefficient and average path length, while the nearest-neighbor coupled network has a large clustering coefficient and average path length. The Kleinberg directed small-world network has a large clustering coefficient and a small average path length. Kleinberg's directed small-world and nearest-neighbor coupled network demonstrate relatively excellent network capacity among these six network types. Fig 16 illustrates

Table 3. Comparison of classic network models.

Network Type	Number of edges	Clustering coefficient	Average path length	Maximum reliable capacity	Subgraph average number of nodes	Subgraph average number of edges	Subgraph average number of WCCs
ER random graph [29]	6070	0.006	3.797	85	128.139	116.708	12.044
Globally coupled network [36]	999000	1.000	1.000	341	60.000	180.000	1.000
Nearest-neighbor coupled [3]	6000	0.600	29.235	576	167.234	241.087	18.768
Star coupled network [37]	1998	0.000	1.996	21	60.900	62.900	1.000
Price network [33]	5978	0.170	85.116	0	0	0	0
Kleinberg network [31]	6995	0.440	4.613	6144	98.320	98.180	24.084

https://doi.org/10.1371/journal.pcsy.0000019.t003

examples of sample storage corresponding to the two networks. A common feature observed in both networks is the presence of numerous small WCCs. As previously mentioned in the network capacity analysis, the essence of large network capacity lies in the arrangement and combination of WCCs. Since both networks have relatively high clustering coefficients, it is quite easy to form small components locally. Each subgraph can be considered a combination of several small components, resulting in a higher network capacity. The reason for the higher capacity of the Kleinberg network is that, due to its lower average path length, it is easier to form some large WCCs. These large WCCs not only exhibit higher distinguishability but also have a higher resource reuse rate for their nodes, thus positively impacting the improvement of network capacity. In contrast, these characteristics are not present in the ER random graph. Due to its low clustering coefficient and average path length, most weakly connected components formed by the ER random graph are large in scale. This is the difference in capacity caused by different network structures.

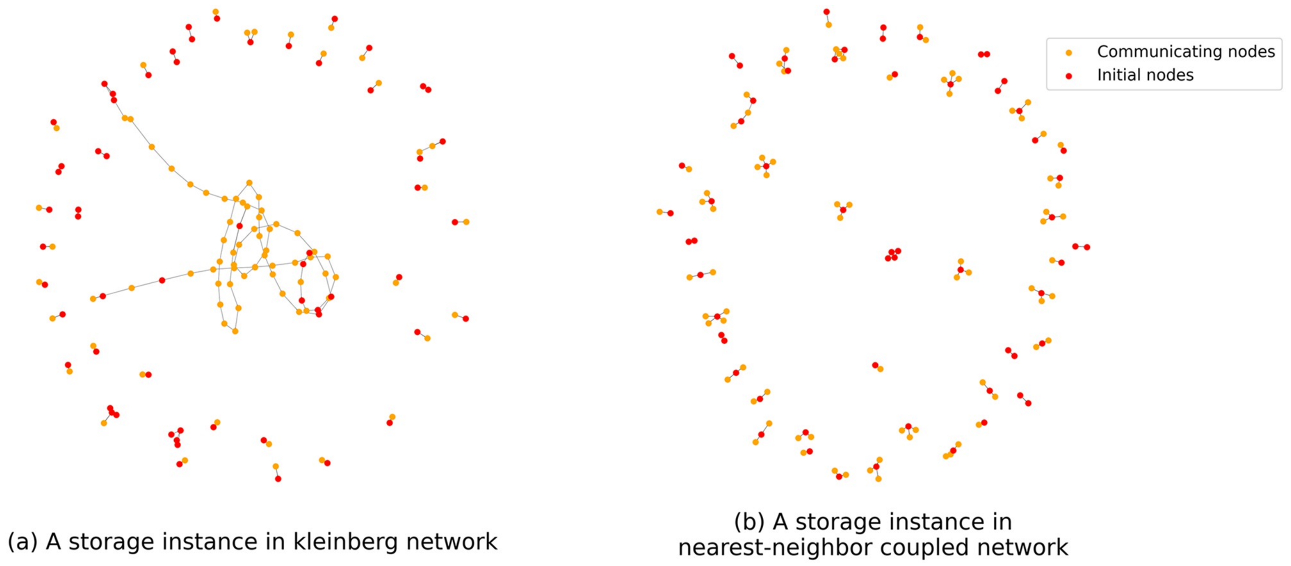

Communicating nodes
Initial nodes
(a) A storage instance in kleinberg network
(b) A storage instance in nearest-neighbor coupled network

Fig 16. Network storage example (a) Kleinberg network storage example. (b) Nearest-neighbor coupled network storage example.

https://doi.org/10.1371/journal.pcsy.0000019.g016

3.5 Performance comparison with other models

This section compares the performance of the model proposed in this paper with other models. There are three models involved in the test: the first one is the memory model based on directed graph storage proposed in this paper, abbreviated as MMDGS. The second one is the discrete Hopfield network model [15]. The third one is the SAM (Sparse Associative Memory) model [38]. The test focuses on the total capacity comparison, and the parameters and updating methods of each network model are as follows.

MMDGS. The network has 500 nodes and 3265 edges (sparse graph). The node index table capacity is 20.

Hopfield. The network has 500 nodes and 249,500 edges (fully connected network). The update strategy is asynchronous, meaning only one node state is updated at each moment.

SAM. The network has 500 nodes, with parameters set according to the original paper. Each memorized sample generates $h = 2$ new nodes in the hidden layer. These h nodes establish excitatory connections with each activated node in the input layer. The activated nodes in the input layer have a $p_s = 0.1$ probability of forming excitatory connections with the newly generated h nodes in the hidden layer. The update rule for each node in the hidden layer is shown in formula (3), where $y_j(t)$ denotes the state of node j in the hidden layer at time t. H is the Heaviside function, satisfying $H(x \geq 0) = 1$. W_{ji} is the connectivity matrix from the input layer to the hidden layer. $\theta = 0.6p_s r$ is the activation threshold, and r is the size of the activation point set in the input layer, i.e., the sample scale. The update rule for each node in the input layer is shown in formula (4), where $x_k(t + 1)$ denotes the state of node k in the input layer at time $t + 1$. U_{kj} is the connectivity matrix from the hidden layer to the input layer. T denotes the total number of stored memory samples.

$$y_j(t) = H(\sum_{i=1}^{n} W_{ji}x_i(t) - \theta) \quad \forall j \tag{3}$$

$$x_k(t+1) = H(\sum_{j=1}^{Th} U_{kj}y_j(t) - 1) \quad \forall k \tag{4}$$

The experiment involves fixing the scale of individual sample nodes and randomly generating sample sets of different sizes. Each network memorizes all the samples in the sample set, recalls them sequentially, and calculates the average accuracy and completeness of recall. The experiment was repeated 10 times, and the results were averaged. The sample node scales include two types: 50 (small scale) and 200 (large scale).

The results of the experiments at small scales are shown in Fig 17. MMDGS performs well in terms of accuracy and completeness, while the Hopfield network performs poorly. This is because Hopfield's network structure is fully connected. Since the sample scale is small, the active nodes receive many inhibitory inputs at each iteration, resulting in an inability to maintain their active state, ultimately leading to lower capacity. SAM exhibits higher completeness of sample recall, with accuracy decreasing as the number of memorized samples increases and stabilizing at around 200. This stabilization value of 0.1 indicates that all nodes in the input layer are activated, which is a meaningless recall.

The results of the experiments at large scales are shown in Fig 18. MMDGS also maintains a very high average accuracy and completeness, indicating that the capacity of MMDGS is substantial. Unlike the small-scale experiments, Hopfield's capacity performs better in large-scale tests, while SAM performs worse. As the sample scale increases, the active nodes in the Hopfield network are less likely to be overwhelmed by inhibitory inputs during iteration. The

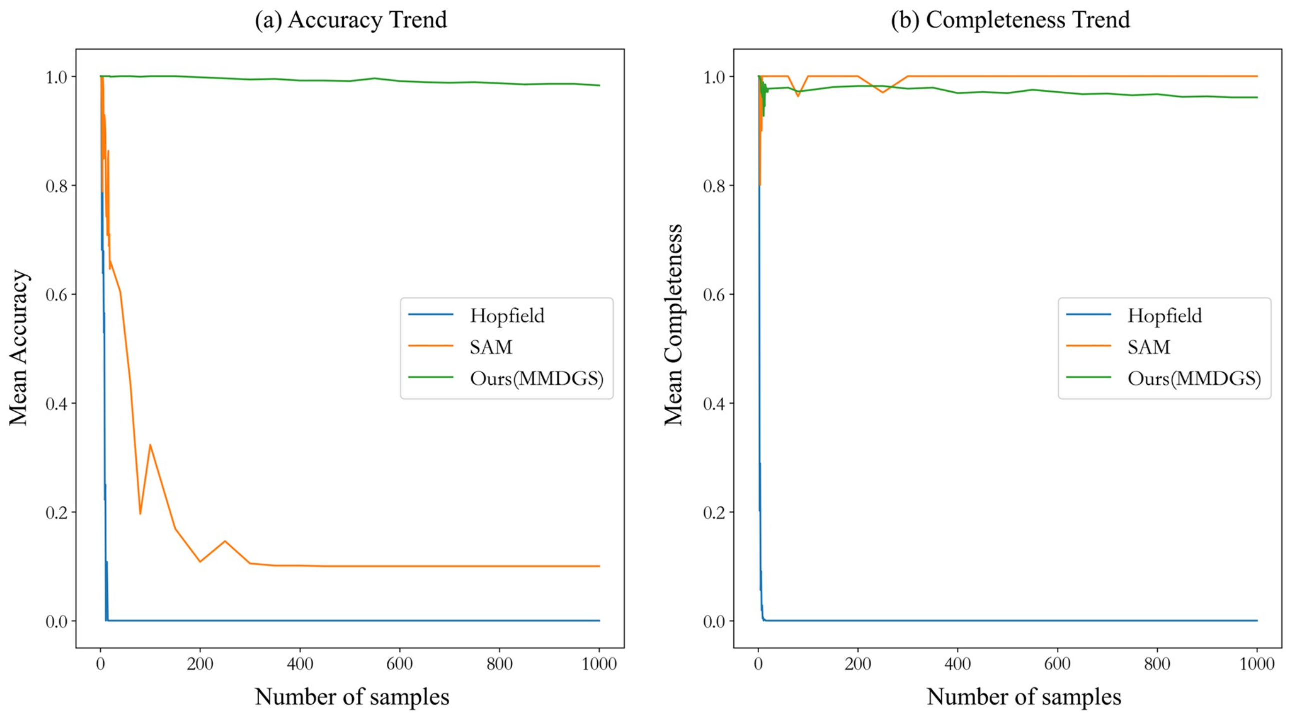


Fig 17. Comparison of the capacities of three models (sample node scale of 50).

https://doi.org/10.1371/journal.pcsy.0000019.g017

network can iterate normally up to local extremes, at which point the capacity increases. For SAM, each sample is mapped to *h* nodes in the hidden layer. Whenever a node in the hidden layer is activated, the corresponding sample is awakened. As the sample scale increases and more samples are memorized, more nodes in the hidden layer are connected to each node in the input layer, increasing the probability of false activation. Therefore, SAM has more capacity at small sample scales. This also causes SAM to depend on the content of the memorized samples. In Fig 18, it can be seen that the recall accuracy of SAM has significant jitter, indicating poor capacity stability.

4 Conclusion

In this paper, we employ subgraphs as physical carriers for information storage and leverage nodes' autonomous adaptive learning behavior to achieve a large-capacity and stable directed graph storage model. The individual nodes' learning behavior does not need a global view, meaning that the tiny algorithms operating within each node do not work under strong central control and are entirely decentralized. Both the learning behavior and the supporting hardware resources are fine-grained and distributed and can, in theory, be highly parallel in physical implementation.

The storage capacity of the network depends on factors such as connectivity and network structure. The dense graph has better connectivity, the subgraphs generated by the samples are usually gathered together, and the communication nodes are rarely used. The measured capacity at this time is low, approaching the theoretical capacity limit that disallows resource reuse. Sparse graphs exhibit poor connectivity, and the sample-generated subgraphs are generally more dispersed, often consisting of several weakly connected components. In this case, the sample-generated subgraphs can be viewed as a permutation of connected components,

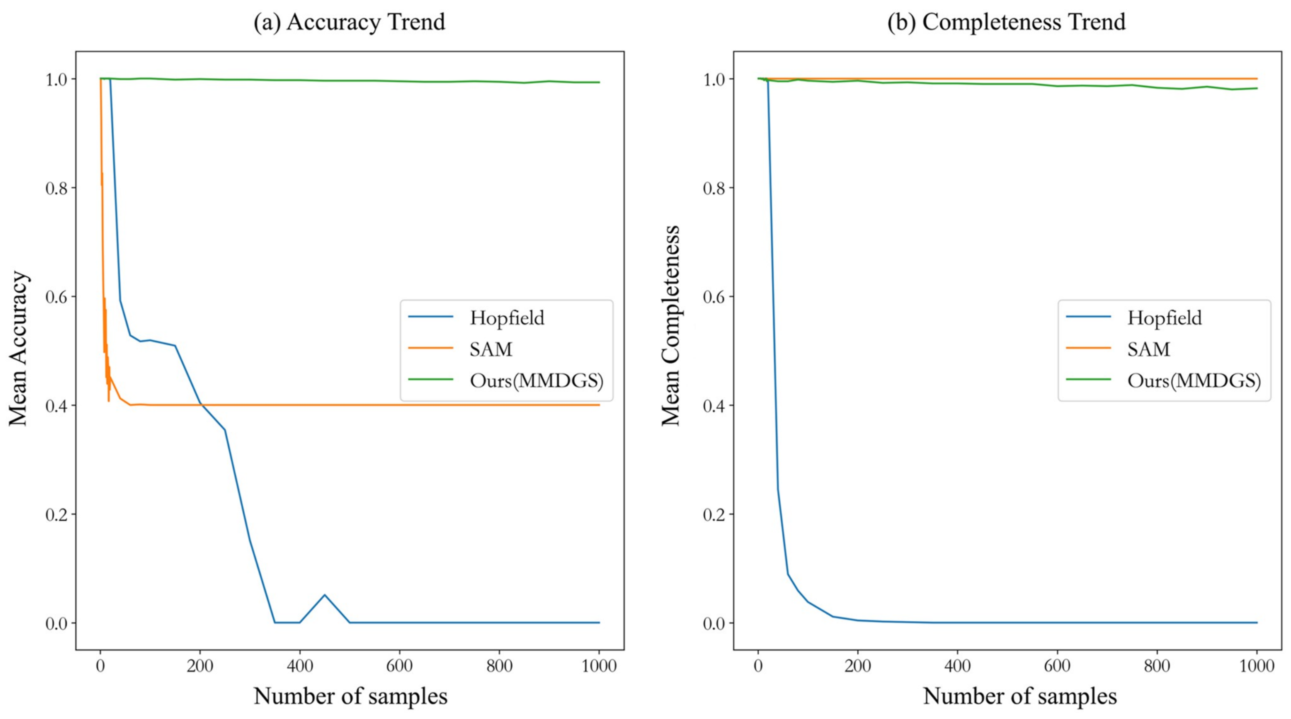


Fig 18. Comparison of the capacities of three models (sample node scale of 200).

https://doi.org/10.1371/journal.pcsy.0000019.g018

significantly increasing the network capacity. Tests have shown that a sparse random directed graph with 500 nodes and 3101 edges can store nearly 8000 memory samples with over 80% accuracy and completeness. In contrast, a dense graph with 500 nodes and 12606 edges can only store around 300 memory samples.

Sparse graphs have fewer resources than dense graphs, but the actual number of samples they can store is tens of times more than dense graphs. It demonstrates that resource abundance is not the sole factor determining network capacity. The network's structural properties, such as connectivity, clustering coefficient, and average path length, are also crucial. Biological neuronal networks exhibit sparse connections and show large capacity and low power consumption characteristics. To some extent, this paper also provides a possible explanation for how biological neuronal networks can achieve memory functions.

The study of memory mechanisms involves interdisciplinary research, such as cognitive psychology, which investigates memory through behavioral experiments, and neurobiology, which explores memory mechanisms at the anatomical and electrophysiological levels. However, many unknown details still exist in linking the microscopic molecular and cellular levels with the macroscopic cognitive and behavioral levels to form a complete information processing mechanism. Currently, there is a lack of sufficient understanding of the biological implementation process of memory mechanisms in the brain, making it challenging to accurately model these mechanisms.

In this paper, we abstract the brain's neuronal network into a directed graph storage model based on neurobiological constraints such as distributed processing, local perspective, and autonomous activity. By combining theories from graph theory and multi-agent systems, we investigate the storage capacity of an active directed graph from the perspective of graph theory. From the algorithmic level, we construct a directed graph storage model that conforms to neurobiological constraints, providing a potential perspective and approach for exploring memory mechanisms.

Author Contributions

Conceptualization: Hui Wei.

Data curation: Hui Wei, Weihua Miao.

Formal analysis: Hui Wei, Weihua Miao.

Funding acquisition: Hui Wei.

Investigation: Hui Wei, Weihua Miao.

Methodology: Hui Wei, Weihua Miao.

Project administration: Hui Wei.

Resources: Hui Wei.

Software: Hui Wei, Fushun Li, Weihua Miao.

Supervision: Hui Wei.

Validation: Hui Wei.

Visualization: Hui Wei, Weihua Miao.

Writing – original draft: Hui Wei, Fushun Li, Weihua Miao.

Writing – review & editing: Fushun Li.

References

1. Kriegeskorte N, Douglas PK. Cognitive computational neuroscience. Nat Neurosci. 2018; 21: 1148–1160. https://doi.org/10.1038/s41593-018-0210-5 PMID: 30127428
2. Sporns O. Graph theory methods: applications in brain networks. Dialogues Clin Neuro. 2018; 20: 111–121. https://doi.org/10.31887/DCNS.2018.20.2/osporns PMID: 30250388
3. Bullmore E, Sporns O. Complex brain networks: graph theoretical analysis of structural and functional systems. Nat Rev Neurosci. 2009; 10: 186–198. https://doi.org/10.1038/nrn2618 PMID: 19190637
4. Meliza CD, Dan Y. Receptive-field modification in rat visual cortex induced by paired visual stimulation and single-cell spiking. Neuron. 2006; 49: 183–189. https://doi.org/10.1016/j.neuron.2005.12.009 PMID: 16423693
5. Yoo M, Yang YS, Rah JC, Choi JH. Different resting membrane potentials in posterior parietal cortex and prefrontal cortex in the view of recurrent synaptic strengths and neural network dynamics. Front Cell Neurosci. 2023; 17: 1153970. https://doi.org/10.3389/fncel.2023.1153970 PMID: 37519632
6. Lovinger DM, Mateo Y, Johnson KA, Engi SA, Antonazzo M, Cheer JF. Local modulation by presynaptic receptors controls neuronal communication and behaviour. Nat Rev Neurosci. 2022; 23: 191–203. https://doi.org/10.1038/s41583-022-00561-0 PMID: 35228740
7. Scholl B, Thomas CI, Ryan MA, Kamasawa N, Fitzpatrick D. Cortical response selectivity derives from strength in numbers of synapses. Nature. 2021; 590: 111–114. https://doi.org/10.1038/s41586-020-03044-3 PMID: 33328635
8. Campagnola L, Seeman SC, Chartrand T, Kim L, Hoggarth A, Gamlin C, et al. Local connectivity and synaptic dynamics in mouse and human neocortex. Science. 2022; 375: eabj5861. https://doi.org/10.1126/science.abj5861 PMID: 35271334
9. Li S, Sheng ZH. Energy matters: presynaptic metabolism and the maintenance of synaptic transmission. Nat Rev Neurosci. 2022; 23: 4–22. https://doi.org/10.1038/s41583-021-00535-8 PMID: 34782781
10. Dalla Costa I, Buchanan CN, Zdradzinski MD, Sahoo PK, Smith TP, Thames E, et al. The functional organization of axonal mRNA transport and translation. Nat Rev Neurosci. 2021; 22: 77–91. https://doi.org/10.1038/s41583-020-00407-7 PMID: 33288912

11. Sagner A, Zhang I, Watson T, Lazaro J, Melchionda M, Briscoe J. A shared transcriptional code orchestrates temporal patterning of the central nervous system. Plos Biol. 2021; 19: e3001450. https://doi.org/10.1371/journal.pbio.3001450 PMID: 34767545

12. Chaudhuri R, Fiete I. Computational principles of memory. Nat Neurosci. 2016; 19: 394–403. https://doi.org/10.1038/nn.4237 PMID: 26906506

13. Dorri A, Kanhere SS, Jurdak R. Multi-agent systems: A survey. IEEE Access. 2018; 6: 28573–28593. https://doi.org/10.1109/ACCESS.2018.2831228

14. Josselyn SA, Tonegawa S. Memory engrams: Recalling the past and imagining the future. Science. 2020; 367: eaaw4325. https://doi.org/10.1126/science.aaw4325 PMID: 31896692

15. Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the national academy of sciences. 1982; 79: 2554–2558. https://doi.org/10.1073/pnas.79.8.2554 PMID: 6953413

16. Krotov D, Hopfield JJ. Dense associative memory for pattern recognition. Advances in neural information processing systems, 2016; 1172–1180.

17. Demircigil M, Heusel J, Löwe M, Upgang S, Vermet F. On a model of associative memory with huge storage capacity. J Stat Phys. 2017; 168: 288–299. https://doi.org/10.1007/s10955-017-1806-y

18. Ramsauer H, Schäfl B, Lehner J, Seidl P, Widrich M, Adler T, et al. Hopfield Networks Is All You Need. ICLR 2021; 2021; 1–94.

19. Kosko B. Bidirectional associative memories. IEEE Transactions on Systems, Man, and Cybernetics. 1988; 18: 49–60. https://doi.org/10.1109/21.87054

20. Kosko B. Bidirectional associative memories: unsupervised Hebbian learning to bidirectional backpropagation. IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2021; 51: 103–115. https://doi.org/10.1109/TSMC.2020.3043249

21. Wei H, Li F. The storage capacity of a directed graph and nodewise autonomous, ubiquitous learning. Front Comput Neurosc. 2023; 17: 1254355. https://doi.org/10.3389/fncom.2023.1254355 PMID: 37927548

22. Senk J, Kriener B, Djurfeldt M, Voges N, Jiang HJ, Schüttler L, et al. Connectivity concepts in neuronal network modeling. Plos Comput Biol. 2022; 18: e1010086. https://doi.org/10.1371/journal.pcbi.1010086 PMID: 36074778

23. Abdou K, Shehata M, Choko K, Nishizono H, Matsuo M, Muramatsu SI, et al. Synapse-specific representation of the identity of overlapping memory engrams. Science. 2018; 360: 1227–1231. https://doi.org/10.1126/science.aat3810 PMID: 29903972

24. Ohkawa N, Saitoh Y, Suzuki A, Tsujimura S, Murayama E, Kosugi S, et al. Artificial association of prestored information to generate a qualitatively new memory. Cell Rep. 2015; 11: 261–269. https://doi.org/10.1016/j.celrep.2015.03.017 PMID: 25843716

25. Tritsch NX, Granger AJ, Sabatini BL. Mechanisms and functions of GABA co-release. Nat Rev Neurosci. 2016; 17: 139–145. https://doi.org/10.1038/nrn.2015.21 PMID: 26865019

26. Chen L, Cummings KA, Mau W, Zaki Y, Dong Z, Rabinowitz S, et al. The role of intrinsic excitability in the evolution of memory: Significance in memory allocation, consolidation, and updating. Neurobiol Learn Mem. 2020; 173: 107266. https://doi.org/10.1016/j.nlm.2020.107266 PMID: 32512183

27. Luo L. Architectures of neuronal circuits. Science. 2021; 373: eabg7285. https://doi.org/10.1126/science.abg7285 PMID: 34516844

28. Manning CD. Introduction to Information Retrieval. Syngress Publishing; 2008.

29. Erdos P, Rényi A. On the evolution of random graphs. Publ Math Inst Hung Acad Sci. 1960; 5: 17–60.

30. Erdos P, Renyi A. On random graphs I. Publ Math Debrecen. 1959; 6: 290–297. https://doi.org/10.5486/PMD.1959.6.3-4.12

31. Easley D, Kleinberg J. Networks, Crowds, and Markets: Reasoning about a Highly Connected World. Cambridge University Press; 2010.

32. Watts DJ, Strogatz SH. Collective dynamics of 'small-world' networks. Nature. 1998; 393: 440–442. https://doi.org/10.1038/30918 PMID: 9623998

33. Price D J D S. Networks of scientific papers: The pattern of bibliographic references indicates the nature of the scientific research front. Science. 1965; 149: 510–515. https://doi.org/10.1126/science.149.3683.510

34. Wang XF, Li X, Chen G R. Network science: an introduction. Higher Education Press; 2012. 95–142.

35. Fagiolo G. Clustering in complex directed networks. Physical Review E—Statistical, Nonlinear, and Soft Matter Physics. 2007; 76: 026107. https://doi.org/10.1103/PhysRevE.76.026107 PMID: 17930104

36. Barabási AL. Network science. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences. 2013; 371: 20120375. https://doi.org/10.1098/rsta.2012.0375 PMID: 23419844

37. Newman M. Networks. Oxford University Press; 2018.

38. Hoffmann H. Sparse associative memory. Neural Comput. 2019; 31: 998–1014. https://doi.org/10.1162/neco_a_01181 PMID: 30883276