IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 31, 2020, accepted September 13, 2020, date of publication September 18, 2020, date of current version October 6, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3024974

Prufer Coding: A Vectorization Method for Undirected Labeled Graph

LIN YANG AND YONGJIE WANG
College of Electronic Engineering, National University of Defense Technology, Hefei 230037, China
Anhui Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, Hefei 230037, China
Corresponding author: Lin Yang (yanglin0815@nudt.edu.cn)

ABSTRACT Prufer algorithm is a powerful method for topology vectorization, but the traditional prufer algorithm method can only encode a rootless labeled tree, and no prior work has studied the method of applying it to the graph vectorization. This paper proposes a vectorization method for undirected labeled graphs based on the prufer algorithm, including graph encoding and decoding algorithms. A particular case was discovered by preliminary experiments, which will reduce the accuracy of the coding algorithm (when the node size reaches more than 150, the accuracy can only reach about 60%), so a connectivity check mechanism that based on the Warshall algorithm is proposed and added to the coding algorithm. A large number of experimental verifications show that the accuracy of the coding algorithm can reach 100% after introducing this mechanism. Then the length of the vector generated by the coding algorithm is analyzed, and the results show that graph vectorization can improve the efficiency of partial topology calculation. Finally, the defects of the algorithm are discussed. The most significant defect is that the length of the vector generated by the encoding algorithm is uncertain, which will prevent it from being applied to more topological calculations.

INDEX TERMS Graph vectorization, topology calculation, prufer algorithm, Warshall algorithm, algorithm design.

I. INTRODUCTION

With the explosive growth of the nodes of the system, the complexity of the connected network has increased. Graph vectorization is introduced to simplify the graph representation, thus further simplify graph topology calculation.

Graph vectorization refers to representing the topology information of a graph as a vector through a graph transformation algorithm. This vector is generally one-dimensional. A graph is a mathematical abstraction that is useful for solving many kinds of problems. If we only need to solve the connectivity problem, we can only discuss the undirected graph model of the system. An undirected graph refers to a graph in which each edge symbolizes an unordered, transitive relationship between two nodes. Such edges are rendered as direct lines or arcs [1]. A vectorized description of the graph will simplify the process of solving problems related to the connectivity of the graph.

Featherstone [2] proposed to use a parent array of undirected graphs to describe the connectivity of the bodies. It uses another two arrays to represent the set of children of related bodies and the set of joints on the path between the related bodies and root. Yazar [3] introduced a one-dimensional vector *Pgraph*, which is used to describe the connection between linear graph theory bodies and branches.

The traditional prufer algorithm is a method of coding and decoding labeled trees, which can be used to vectorize unrooted trees, and it was first proposed by Heinz Prufer in 1918 when he proved Cayley's theorem.

Prufer sequence is not incredibly widely used, but it can be applied on some special occasions, such as be integrated into the design of the Genetic Algorithm (GA) [4]–[6] and used to solve the Minimum Spanning Tree (MST) problem [7]. Reference [8] proposes a new XML schema matching framework based on the use of prufer encoding to improve the performance of identifying and discovering complex matches. Reference [9] uses the prufer code to define martingale of the tree and establish a concentration result for a specific family of functions over random trees with given degrees. Reference [10] designed a BMEP polytope iterative enumeration algorithm based on the Prufer coding method of rootless label tree, combined with the multi-faceted combination algorithm

The associate editor coordinating the review of this manuscript and approving it for publication was Haipeng Yao.

of the Balanced Minimum Evolution Problem (BEMP). The chain structure of the objective supply chain [11] and the branched polymer can be represented as a tree structure. Reference [12] generates random trees with the same degree distribution by repeatedly modifying the prufer code to create the randomly branched polymers. Reference [13] used the prufer algorithm to encode its proposed skeleton graph model and check the isomorphism of the skeleton graph based on the prufer sequence, but the skeleton graph proposed in reference [13] is actually a tree structure.

Compared with the tree structure, the graph structure (or the mesh structure) has a broader range of application scenarios. Can the random tree generation method based on the prufer algorithm proposed in reference [12] be extended to the generation of random graphs? Whether the graph isomorphism detection method based on the prufer algorithm in reference [13] could be accurately applied to graph structures? Although scholars have proposed many improved methods for the traditional prufer algorithm, no one has considered applying it to graph vectorization. In order to realize it, our contributions could be summarized as follows:

a) Propose a method for undirected labeled graph vectorization based on the prufer algorithm, including graph encoding and decoding algorithm.

b) Propose a method to check the connectivity of the graph based on the Warshall algorithm and introduce an improved approach, then apply it to increase the accuracy of the prufer algorithm in coding and decoding undirected labeled graph.

Finally, the application prospect of the graph vectorization in topology calculation will be analyzed. The process block diagram of the topology vectorization is shown in Figure 1.

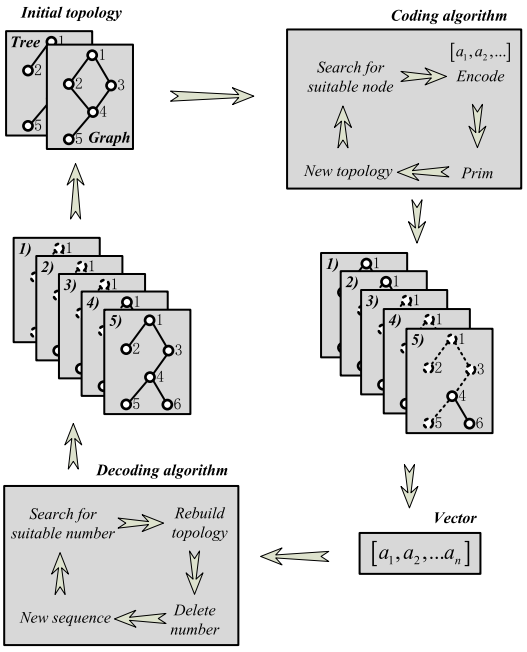

FIGURE 1. Process block diagram of topological vectorization.

II. REVIEW OF THE PRUFER ALGORITHM

A. PRUFER CODING OF ROOTLESS LABELED TREE

Prufer sequence encoding refers to converting a tree into a character string, and decoding refers to converting a character string into a tree [14].

First, briefly introduce the prufer coding process of rootless trees: Let T be a tree with n vertices; then tree T is called a labeled tree if the n vertices are distinguished from one another by names such as $v_1, v_2, \ldots, v_n$ [15]. Assuming that the known n vertices are simply marked as $1, 2, \ldots, n$, then suppose that T is one of the trees, and the node with the smallest label in the leaves is a_1, its adjacent node is b_1. When the point a_1 and the edge (a_1, b_1) are trimmed from the graph, the point b_1 becomes the leaf of the remaining tree T_1. Then search the leaf with the smallest label in the remaining tree T_1, set to a_2, the adjacency point of a_2 is b_2, and trim a_2 and edge (a_2, b_2) from T_1. Continue this step *n-2* times until there is one edge left. Then tree T can be expressed as the sequence $b_1, b_2, \ldots, b_{n-2}$, which is called the prufer sequence, and this process is called the prufer coding algorithm. The coding steps are summarized as follows [16]:

step_1: Cut the leaf nodes and edges in order from small to large according to vertex labels.

step_2: Record the node number that connected to the leaf node on the trimmed edge.

step_3: Repeat step_1 and step_2 until only two nodes and edges between them are left in the tree, the algorithm is end.

The following is a concrete example to illustrate the rootless tree coding and decoding method of the Prufer sequence. The constructed rootless tree is shown in Figure 2.

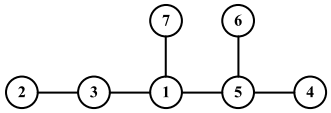

FIGURE 2. A rootless tree composed of 7 nodes.

Firstly, according to the coding step_1, node 2 and the edge (2, 3) with the smallest sequence number among the leaf nodes are cut out to generate a new tree.

According to the coding step_2, record the node number 3 adjacent to node 2, so the current prufer sequence is 3.

Then according to the coding step_3, the above process is repeated until one edge remains. The entire process is shown in Figure 3.

The prufer sequence changes as follows.

$$[3] \rightarrow [3, 1] \rightarrow [3, 1, 5] \rightarrow [3, 1, 5, 5] \rightarrow [3, 1, 5, 5, 1]$$

Finally, the prufer sequence is [3, 1, 5, 5, 1].

B. PRUFER DECODING OF ROOTLESS LABELED TREE

Provide two sequences $1, 2, \ldots, n$ and $b_1, b_2, \ldots, b_{n-2}$, which are sequential sequence (*SeqtSeq*) and prufer sequence (*PruferSeq*), respectively. The tree T can be conversely decoded from $b_1, b_2, \ldots, b_{n-2}$.

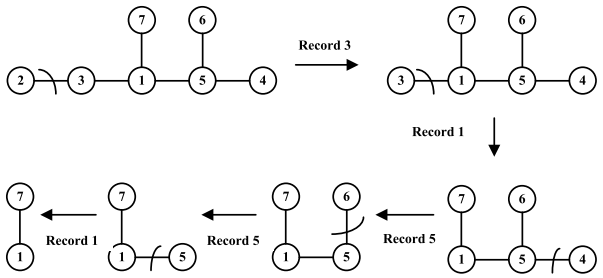

FIGURE 3. Prufer coding process of tree T.

In above coding process, since a_1 has been cut from the tree T when recording b_1, a_1 will not appear in $b_2, \ldots, b_{n-2}$, so find the first number that does not appear in *PruferSeq* from *SeqtSeq*. This number is obviously a_1, and at the same time, rebuild the edge (a_1, b_1), then eliminate a_1 from *SeqtSeq* and eliminate b_1 from *PruferSeq*. Continue the above steps $n-2$ times until the *PruferSeq* becomes an empty set. At this time, the *SeqtSeq* will have two numbers a_k, a_j left, and the edge (a_k, a_j) will be the last edge of the tree T. Decoding steps are as follows:

step_1: Construct the *SeqtSeq* according to the node number of the tree. Find the number that is not in *PruferSeq* and is located on the leftmost side of the *SeqtSeq*. Connect it to the leftmost number of the *SeqtSeq* to rebuild this edge.

step_2: After completing the step_1, the two node numbers of *SeqtSeq* and *PruferSeq* are eliminated to form two new sequence.

step_3: Repeat step_1 and step_2 several times until only two numbers left in the *SeqtSeq*. Then rebuild the edge corresponding to the remaining two numbers, and the algorithm terminates.

Continue take the above tree T as an example. The *PruferSeq* that we get is [3,1,5,5,1], according to step_1, construct the *SeqtSeq*: [1,2,3,4,5,6,7], the leftmost sequence number that in *SeqtSeq* but not in *PruferSeq* is 2, and the leftmost number of the *SeqtSeq* is 3, so rebuild edge (2,3), as shown in Figure 4. According to the decoding step_2, delete the number 2 in *SeqtSeq* and the leftmost number 3 in *PruferSeq*, here we expressed it as [1,②,3,4,5,6,7] and [③,1,5,5,1]. Get the new *SeqtSeq*: [1,3,4,5,6,7] and *PruferSeq*: [1,5,5,1].

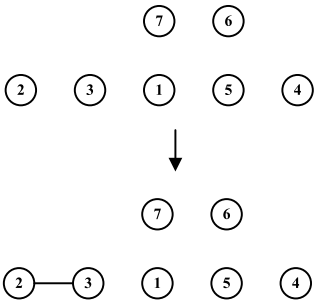

FIGURE 4. Decoding side (2, 3).

According to the decoding step_3, the above processes will be repeated until *SeqtSeq* has only two numbers left: [1,7], finally rebuild the edge (1,7). The changes of *SeqtSeq* and *PruferSeq* are shown below.

$$\begin{cases} [1, ③, 4, 5, 6, 7] \\ [①, 5, 5, 1] \end{cases} \rightarrow \begin{cases} [1, ④, 5, 6, 7] \\ [⑤, 5, 1] \end{cases} \rightarrow \begin{cases} [1, 5, ⑥, 7] \\ [⑤, 1] \end{cases}$$
$$\rightarrow \begin{cases} [1, ⑤, 7] \\ [①] \end{cases} \rightarrow \begin{cases} [1, 7] \\ \phi \end{cases}$$

Edges (3,1), (4,5), (6,5), (5,1), (1,7) will be decoded in order, as shown in Figure 5.

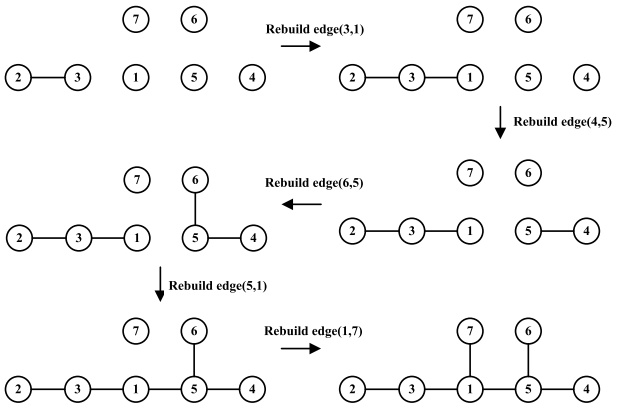

FIGURE 5. The decoding process of the rootless tree.

Finally, we get a tree that is the same as the rootless tree in Figure 2. The prufer coding and decoding algorithm processes are shown in Figure 6.

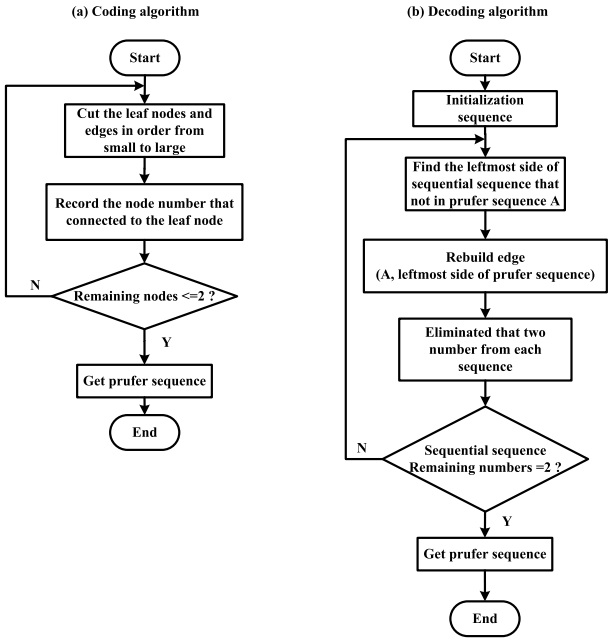

FIGURE 6. Algorithm flowchart of rootless labeled tree.

Scholars have optimized the prufer encoding and decoding algorithms, and have proposed many improved algorithms. Reference [16] and [17] propose linear time algorithms. Using the integer sorting algorithm obtained by the particularity of the integer values to be sorted, the prufer encoding and decoding problems are simplified to integer sorting problems, which can better improve the efficiency of rootless tree prufer coding and decoding. Reference [18] uses simple arrays to improve prufer algorithm, which can improve the time complexity of prufer coding to $O(n)$. Reference [14] studied a decoding algorithm that scanned the prufer sequence in reverse order and proved that the algorithm could run in linear time without the need for additional data structures or sorting processes.

III. PRUFER ALGORITHM FOR UNDIRECTED LABELED GRAPH

Traditional prufer algorithms can be used to encode and decode a rootless labeled tree. However, compared to a rootless tree, graphs are more widely used to solve network problems, so it is necessary to design a method for coding and decoding labeled graphs.

The tree and graph are both non-linear data structures, but the graph is more abstract and complex than a tree. Compared with a tree, the graph has a unique structure, which is named cycle. Graph coding needs to focus on solving the coding problem of the cycle. In order to emphasize the structural nature of the graph, we only discuss the coding and decoding of the undirected simple graph, which does not include parallel edges and self-loops. Figure 7 shows a simple undirected labeled graph with a cycle structure and leaf node.

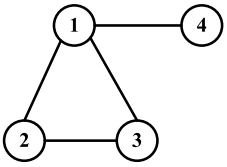

FIGURE 7. A simple undirected labeled graph G.

By coding graph G, we will find some problems: if use the prufer coding method of the rootless tree to coding graph G, the node 4 and edge (1,4) in graph G will be trimmed first. Then the remaining nodes 1, 2, and 3 form a cycle, where there are no more leaf nodes. In this situation, which node and which edge should be trimmed next? In order to successfully coding the undirected labeled graph, we need to find a suitable way to solve this problem.

A. PRUFER CODING OF UNDIRECTED LABELED GRAPH

First of all, the single node cropping rules are specified: Each cropping step trim the node and all edges connected to it. In the prufer coding algorithm of the rootless tree, the clipped node is always leaf-node; there are only one adjacent node that needs to be recorded each time. Therefore, the single node recording rule is specified: Recording all adjacent nodes of the clipped node in order, then record that clipped node at the end.

Suppose that the n vertex of the undirected labeled graph G is denoted as $a_1, a_2, \ldots, a_n$. The coding steps are designed as follows:

step_1: If the current undirected labeled graph has leaf nodes $a_i, \ldots, a_j$, cut out the smallest node a_{min} among the leaf nodes, as well as the edge (a_i, b_i) formed with the adjacent node b_i. If there is no leaf node left, the one with the smallest sequence number among the remaining nodes will be trimmed.

step_2: If the clipped node is a leaf node, only its adjacent node b_i should be recorded; if the clipped node is not a leaf node and its degree is j(j $\geq$ 2), All nodes $b_1, b_2, \ldots, b_j$ that connected to a_i through edges $(a_i, b_1), (a_i, b_2), \ldots, (a_i, b_j)$ should be recorded, assuming that $b_1 < b_2 < \ldots < b_j$ follow the order from small to large, record all of them and add the clipped node a_i at the end to generate a sequence $[b_1, b_2, \ldots, b_j, a_i]$.

step_3: When each trim is complete, a new undirected labeled graph will be generated. Continue to repeat step_1 and step_2 until the undirected labeled graph has only two nodes left, and the algorithm terminates.

Taking graph G (Figure 7) as an example graph, its coding process is shown in Figure 8.

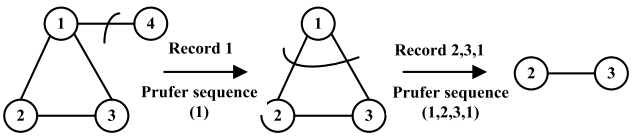

FIGURE 8. The coding process of an undirected labeled graph.

B. PRUFER DECODING OF UNDIRECTED LABELED GRAPH

The decoding of an undirected labeled graph is the reverse process of encoding, so we should correctly restore all detail in the coding process. Due to the complexity of the coding process, several problems should be considered. Firstly, how to rebuild the cycle structure? We know that cycle structure is the particularity of the graph. In the coding algorithm, we recorded all the adjacent nodes of the clipped node and recorded that node at the end. Therefore, in the decoding process, we only need to locate that node in the *PruferSeq*, then rebuild all the adjacent edges with the number in front of it. Secondly, according to the coding algorithm of the rootless labeled tree, the way to rebuild leaf node could adopt the same method, that is, find the leftmost number that included in *SeqtSeq* not appear in *PruferSeq*, and link it to the leftmost number of *PruferSeq* to rebuild that edge.

Decoding steps could be designed as follows.

step_1. Find the number a that located in the leftmost side of *SeqtSeq* but does not exist in the *PruferSeq*.

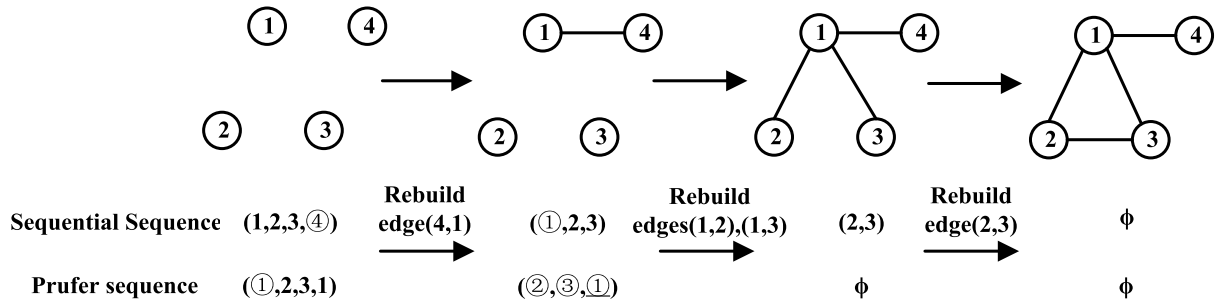

FIGURE 9. Prufer decoding process of undirected labeled graph G.

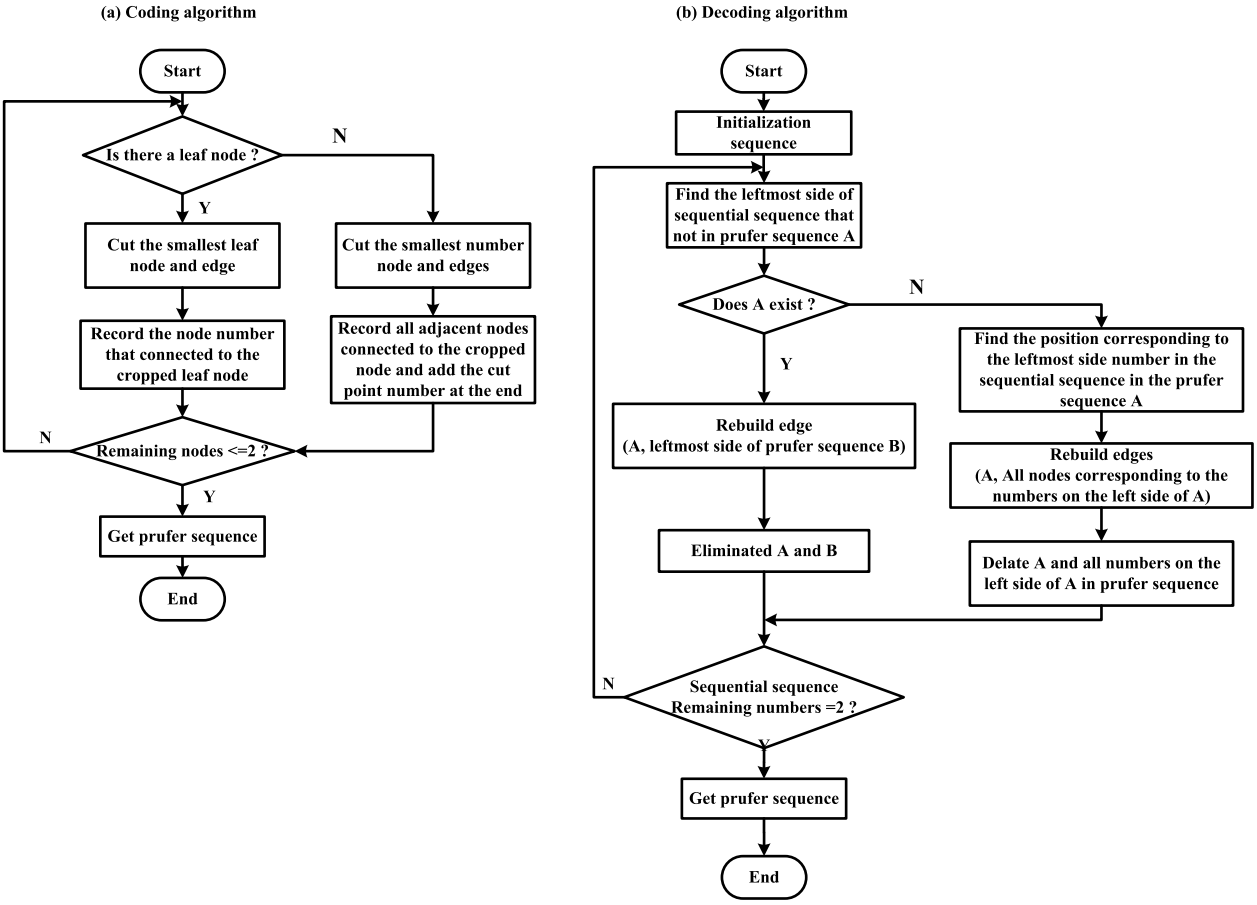

FIGURE 10. Prufer coding and decoding flowcharts for undirected labeled graph.

step_2. Connect the node a with the node b that located in the leftmost side of *PruferSeq*, rebuild edge (a, b).

step_3. Delete a in *SeqtSeq* and b in *PruferSeq*.

step_4. If the above number a does not exist, find the position of the leftmost side number of *SeqtSeq* in *PruferSeq*, mark that number as b_j.

step_5. Connect the node b_j with each node $(b_1, \ldots, b_{j-1})$ that in front of b_j, in order to rebuild edges $(b_j, b_1), \ldots, (b_j, b_{j-1})$.

step_6. Delete number b_j in *SeqtSeq* and all numbers $b_1, \ldots, b_{j-1}, b_j$ in *PruferSeq*;

step_7. Repeat the above process until there are only two numbers left in *SeqtSeq*. Connect the remaining two numbers, rebuild the final edge, the algorithm is over.

Taking the undirected labeled graph G as an example, we have got the *PruferSeq* in above. According to the *PruferSeq*, the decoding process is shown in Figure 9.

The prufer coding and decoding flow for the undirected labeled graph is shown in Figure 10. To facilitate experimental verification, the pseudocode of algorithm is designed as follows:

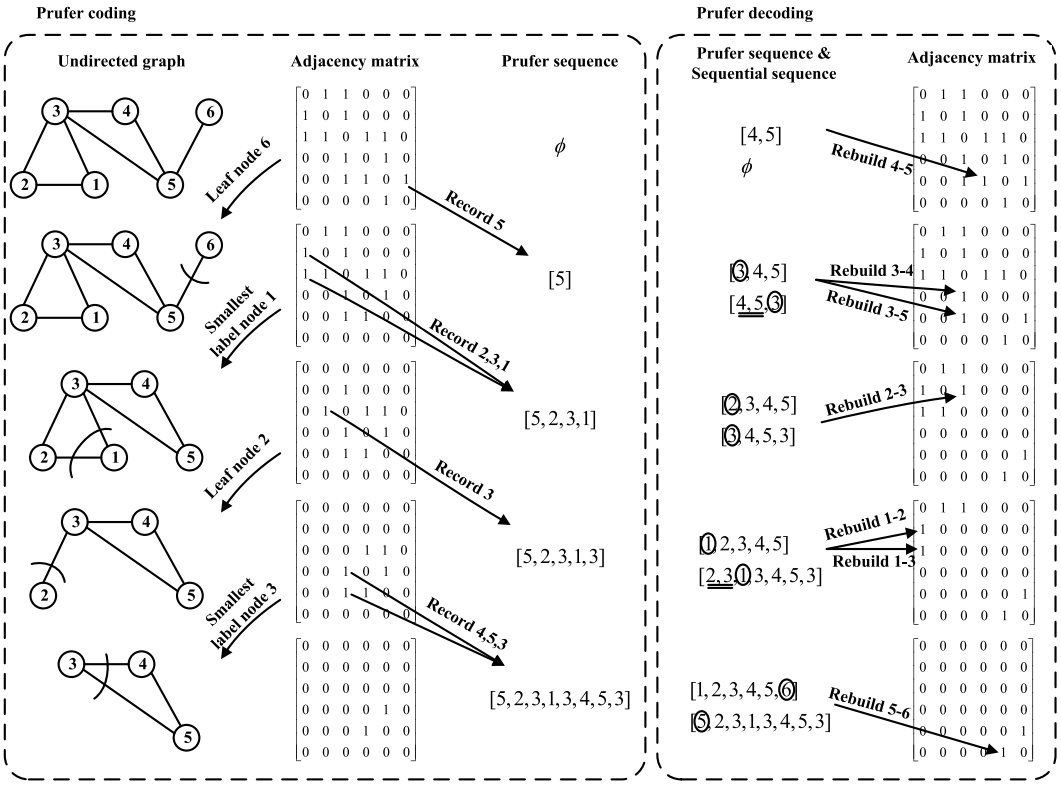

FIGURE 11. Schematic diagram of the experimental codec process.

Algorithm 1 Coding Algorithm

Input:
undirected labeled graph G

Output:
prufer coding sequence *PruferSeq*.

algorithm *PruferCoding* (G)
while $G.nodenum > 2$
  $CurrentNode \leftarrow FindLeafNode\ (G)$
  if $CurrentNode$!= **NULL**
    $AdjNode \leftarrow FindAdjNode\ (CurrentNode)$
    $PruferSeq \leftarrow Record\ (PruferSeq, AdjNode)$
  else $CurrentNode \leftarrow FindSmallestNode\ (G)$
    $Adj \leftarrow FindAdjNode\ (CurrentNode)$
    $PruferSeq \leftarrow Record\ (PruferSeq, Adj, CurrentNode)$
  end if
end while
Prun (*CurrentNode*)
$PruferSeq \leftarrow Record\ (PruferSeq,$ NodeNumof(G))
return *PruferSeq*
end algorithm

In order to clearly describe the algorithm execution process, we intuitively take an undirected labeled graph composed of 6 nodes as an example to show the entire process of its coding and decoding, as shown in Figure 11.

The basic operation of the coding algorithm is to determine whether a node is a leaf node; its time complexity is $O(n)$. Outer while loop needs to determine the remaining nodes number, its time complexity is also $O(n)$, so the time complexity of the coding algorithm is $O(n^2)$. The basic operation of the decoding algorithm is to find the node number that in the leftmost of *SeqtSeq* but not in *PruferSeq*, its time complexity is $O(n^2)$, consider the outer while loop, the time complexity of decoding algorithm is $O(n^3)$.

The optimal time complexity of basic operation can reach $O(n \log n)$. Meanwhile, the time complexity of the outer while loop can be reduced to $O(\log n)$ by selecting the appropriate data storage structure [19], so the optimal coding and decoding algorithm time complexity are $O(n \log n)$ and $O(n \log^2 n)$ respectively.

For undirected labeled graphs with different node size scales, a large of experiments have been carried outs. The accuracy rate of the codec still has not reached 100%, as shown in Table 1.

According to the algorithm execution process, to analyze the causes of algorithm errors, we found that the original graph will be divided into two or more graphs in some particular cases. In this situation, the algorithm execution result will be wrong, as shown in Figure 12.

In such a situation, the node with the small label happens to be the bridge node connecting the two subgraphs, and currently, there is no leaf node. If we trim such a node,

Algorithm 2 Decoding Algorithm

Input:
prufer coding sequence *PruferSeq*
Output:
undirected labeled graph *G*.

```
1: algorithm PruferDecoding (PruferSeq)
2: CreateSeqtSeq (PruferSeq)
3: while SeqtSeq.length > 2
4:    for i ← 1 to sizeof (SeqtSeq)
5:       if NotBelongtoPruferSeqt (SeqtSeq (i))
6:          CurrentNum ← SeqtSeq (i)
7:          break
8:       end if
9:    end for
10:   if CurrentNum != NULL
11:      G ← RebuildEdge (G, CurrentNum, PruferSeq (1))
12:      Eliminate (CurrentNum, PruferSeq (1))
13:   else CurrentNum ← FindinPruferSeqt (SeqtSeq (1))
14:      Adj ← AllLeftNum (CurrentNum)
15:      G ← RebuildEdge (G, Adj CurrentNum);
16:      Eliminate (Adj, CurrentNum)
17:   end if
18: end while
19: G ← Rebuild (G, SeqtSeq)
20: return G
21: end algorithm
```

TABLE 1. Accuracy data of the first experiment.

Size	Result	Count	Percent
[3,21]	Correct	17	5.67%
	Error	283	94.33%
[22,51]	Correct	159	14.52%
	Error	27	85.48%

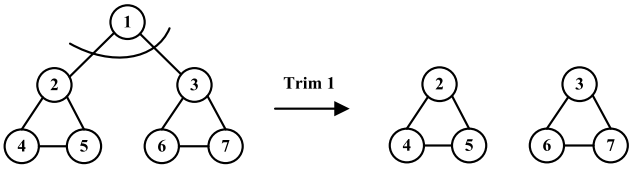

FIGURE 12. The particular case of trimming into two pictures.

the original graph will be divided into two graphs. In order to solve this problem, the shearing condition needs to be added.

Such a problem certainly does not occur when cutting leaf nodes, so it is necessary to detect whether the current undirected graph will be decomposed into multiple graphs in the second case (cutting non-leaf nodes).

The Warshall algorithm uses the idea of dynamic programming to find transitive closures, which can be used to judge the connectivity of the graph [20]. If only need to judge the connectivity of the undirected graph simply, a vector can be introduced to record the reachability of a single node. We know that the *n* power of the adjacency matrix represents the number of paths that each node can reach through *n* hops to another node (including itself), so the connectivity detection algorithm can be designed as follows:

Algorithm 3 Connectivity Check

Input:
undirected labeled Graph G
Output:
connectivity judgment result

```
1: algorithm ConnectionCheck (G)
2: CheckLine ← G (1)
/*Use the first row of the adjacency matrix for inspection*/
3: for PowerCount ← 2 to G.nodenum
4:    CheckLine ← CheckLine and G (1)
/*CheckLine performs AND operation with the first row of
the current matrix*/
5:    if AllOnes (CheckLine)
/*If CheckLine is all 1 then return true*/
6:       return TRUE
7:    end if
8:    G ← G * G_Init; // Continue multiplication
9: end for
10: return FALSE
11: end algorithm
```

If *CheckLine* becomes an all-one array, it means the node that we marked can reach any other nodes, that is, this undirected graph is connected. When considering this particular case, it means that there is no leaf node at present, so a cycle will appear, it will accelerate the check. Only in the worst case, the outer loop needs to be performed *n-2* times. The inner layer is to check *CheckLine*. If we mark the value that has been changed to 1, so that each time only need to check the value that is still 0 in the previous round, the time complexity can be reduced to $O(\log n)$, so the time complexity of this check algorithm is $O(n \log n)$.

Therefore, the algorithm needs to make the following improvements: If it is found that trimming the current non-leaf node will divide the original graph into multiple graphs, then mark and skip this node until a node that does not decompose the original graph is found, exchange it with the smallest marked node, and record this exchange in order to recover when decoding. We introduce a table structure for recording this exchange and return it at the end of the prufer coding algorithm, and meanwhile, it as the input of the prufer decoding algorithm to help restore this exchange.

The improved part can be described as follows:

IV. ALGORITHM APPLICATION

The algorithm proposed in this paper can better implement the vectorization of the undirected labeled graph and record the connectivity of it. Recording the two-dimensional adjacency matrix as a one-dimensional vector, can sometimes

Algorithm 4 Improved Coding Algorithm

◀/*rest of the algorithm*/
if there is no leaf nodes
  $a \leftarrow G.smallestNum$
  Trim (a)
  if *ConnectionCheck* $(G) =$ **FALSE**
    Restore (G)
    MarkNodeSet $\leftarrow a$
  else *ConnectionCheck* $(G) =$ **TRUE**
    if *isempty*(*MarkNodeSet*) $=$ **FALSE**
      ExgTable $\leftarrow$ *Record* (*MarkNodeSet.smallest*, a)
      Clear(*MarkNodeSet*)
    end if
  end if
end if /*rest of the algorithm*/▶
return *ExgTable*

Algorithm 5 Improved Decoding Algorithm

◀/*rest of the algorithm*/
if there is no qualified number
  if *isexchanged* (*SeqtSeq*(1)) $=$ **TRUE**
    $b \leftarrow$ *Recover*(*ExgTable*, *SeqtSeq*(1))
  else
    $b \leftarrow$ *SeqtSeq*(1)
  end if
end if /*rest of the algorithm*/▶

greatly simplify some graph operations and improve the efficiency of solving graph problems, such as the graph isomorphism judgment problem, as shown in Figure 13.

Graph isomorphism is the most rigorous form of exact graph matching, holding all the mapping, which must be a bijection in both directions [21]. *Graph_1* and *Graph_2* in figure 13 are isomorphic because their adjacency matrixes are exactly the same. The prufer sequences obtained according to the algorithm proposed in this paper are also the same: [3,4,1,5,2], and they have the uniqueness of decoding. If the graph isomorphism analysis is performed based on the adjacency matrix, it will take 10 comparison operations, and based on the prufer sequence, it will only require 5 comparison operations so that the efficiency will be sharply improved.

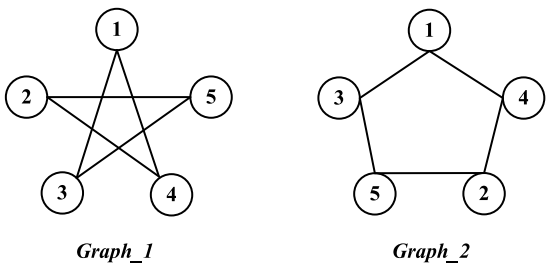

FIGURE 13. Graph isomorphism.

When use the adjacency matrix to store a simple undirected graph, the useful information is distributed in the upper triangle of that matrix, as shown in Figure 14.

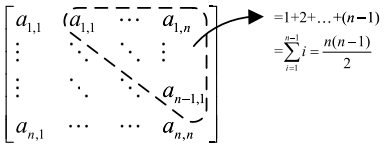

FIGURE 14. Useful information distribution.

The length of prufer sequence is related to the connectivity of the undirected graph, as cropping rules make sure that it will always record by the cropped edges, a conclusion could be made as follow:

$$PruferLen = Edge + CroppedRing$$

Therefore, when the undirected graph is sparse, the space occupied by the prufer sequence to store useful information is always sharply less than $n(n-1)/2$, it will be verified in the later experiment.

V. EXPERIMENT

A. ALGORITHM ACCURACY VERIFICATION

In order to verify the effect of the improvement method, more experiments were carried out.

To compare the accuracy difference between the original algorithm and the improved algorithm, we conducted 500 experiments each for undirected labeled graphs with different numbers of nodes. Table 2 shows the accuracy

TABLE 2. Comparison of two algorithm experiments.

Size	Original	Percent	Improved	Percent
5	500/500	100%	500/500	100%
10	480/500	96%	500/500	100%
15	465/500	93%	500/500	100%
20	434/500	86.8%	500/500	100%
25	438/500	87.6%	500/500	100%
30	431/500	86.2%	500/500	100%
35	418/500	83.6%	500/500	100%
40	407/500	81.4%	500/500	100%
45	399/500	79.8%	500/500	100%
50	391/500	78.2%	500/500	100%
55	385/500	77%	500/500	100%
60	374/500	74.8%	500/500	100%
65	369/500	73.8%	500/500	100%
70	375/500	75%	500/500	100%
75	369/500	73.8%	500/500	100%
80	375/500	75%	500/500	100%
85	363/500	72.6%	500/500	100%
90	348/500	69.6%	500/500	100%
95	351/500	70.2%	500/500	100%
100	378/500	75.6%	500/500	100%

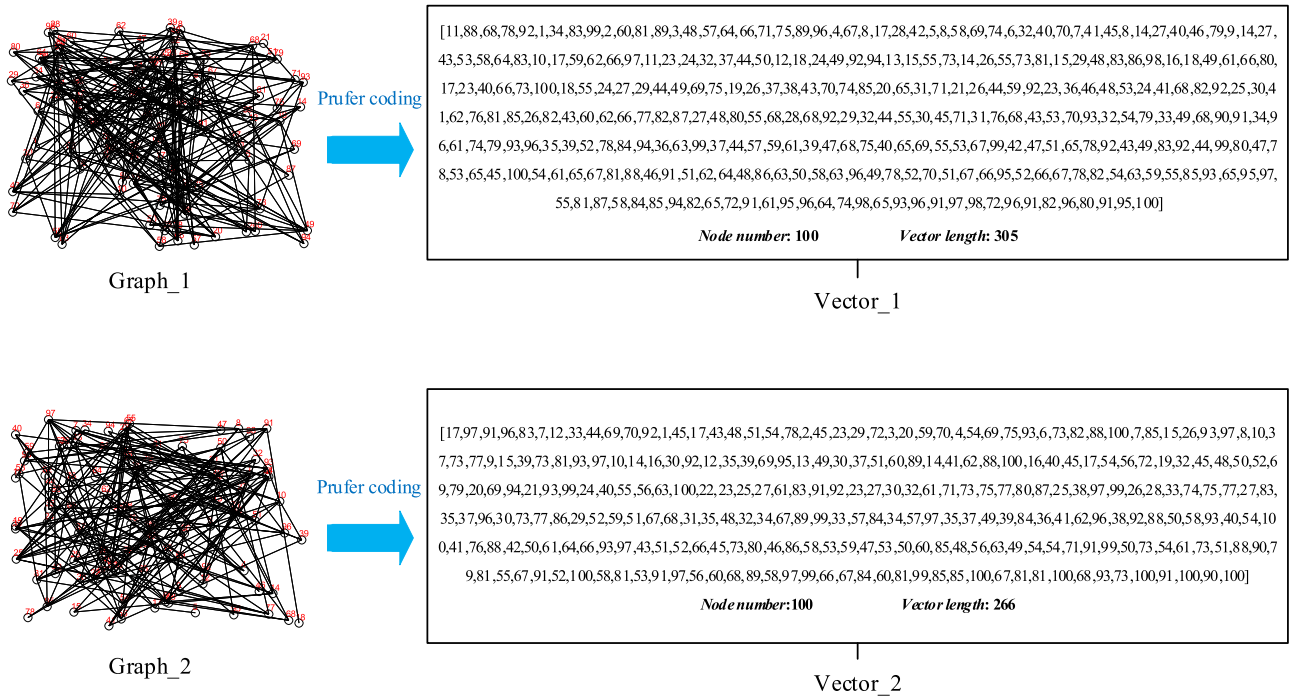

FIGURE 15. Vectors generated by prufer coding algorithm of two undirected labeled graphs with 100 nodes.

comparison of the two algorithm experiments on different *Graph_Size*. The experimental results show that the improved algorithm can always achieve 100% coding and decoding accuracy.

Meanwhile, from the experiments, we can find that the accuracy of the original algorithm will decrease when the node number increases, as the expansion of the graph will increase the probability of the above special case. Although the improvement of the algorithm increases the time cost, it dramatically improves the algorithm accuracy. The accuracy of the original algorithm in large-scale (between 100 to 300) graph is shown in Table 3.

Excluding the influence of random errors, the accuracy rate of the algorithm that does not introduce the connectivity check mechanism will be reduced to less than 60% when the size comes to about 250 nodes.

The main disadvantage of the algorithm is that for different topological graph models, the length of vectors generated by the coding algorithm are not the same. If the vectors can be determined to have the same length, then multiple vectors can be formed into a full matrix (full matrix here refers to needn't to fill in irrelevant information to make the matrix aligned). When using the matrix method to solve topological calculation problems such as subgraph isomorphism, matrix operations can significantly improve the operation efficiency.

Taking two random undirected labeled graphs with 100 nodes as an example, the vectors generated by the algorithm proposed in this paper are shown in Figure 15.

Figure 16 shows two experiments that the topological size from 5 to 100, taking the size step as 5, and performing 50 times coding on each size to generate the average length of the vector.

TABLE 3. The accuracy of original algorithm in large-scale.

Size	Percent	Size	Percent
100	74%	105	78%
110	71%	115	79%
120	64%	125	66%
130	74%	135	66%
140	77%	145	70%
150	68%	155	65%
160	68%	165	53%
170	64%	175	60%
180	64%	185	59%
190	68%	195	67%
200	65%	205	65%
210	65%	215	54%
220	70%	225	64%
230	63%	235	66%
240	59%	245	64%
250	65%	255	62%
260	55%	265	64%
270	55%	275	59%
280	57%	285	64%
290	65%	295	61%
300	53%		

Through data fitting, the power function is used as the fitting model, and the fitting function obtained is approximately as $l = n^{1.237}$, which is better than $n(n-1)/2$.

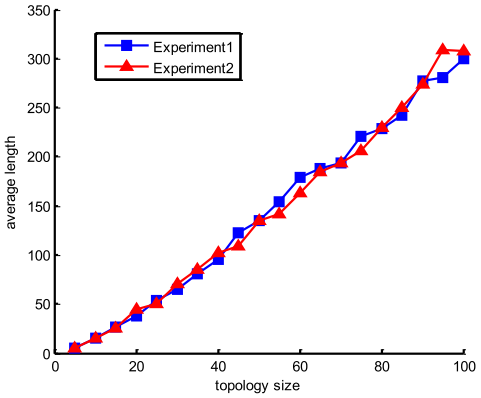

FIGURE 16. The effect of topological size on vector length.

VI. CONCLUSION

This paper has discussed the method of graph vectorization, the prufer coding and decoding algorithms for undirected labeled graph were proposed, and the algorithm was analyzed and improved according to the experimental results. The final experimental results showed that the algorithm could well encode and decode the undirected labeled graph. By analyzing the time complexity of the algorithm, it has acceptable time complexity. The algorithm has a good application scenario, such as being used to generate graphs that meet certain conditions randomly. Besides, the algorithm provides an idea for the vectorization of graphs, which can simplify some graph operations. However, the length of the vector generated by the algorithm cannot be determined, so it cannot be well applied for some specific graph analysis. In the subsequent research, it will continue to explore the applicable range of the method.

REFERENCES

[1] A. B. Sadavare and R. V. Kulkarni, "A review of application of graph theory for network," *Int. J. Comput. Sci. Inf. Technol.*, vol. 3, no. 6, pp. 5296–5300, 2012.
[2] R. Featherstone, "A beginner's guide to 6-D vectors," *IEEE Robot. Automat. Mag.*, vol. 17, no. 4, pp. 88–99, Dec. 2010.
[3] M. N. Yazar and S. M. Yesiloglu, "Path defined directed graph vector (Pgraph) method for multibody dynamics," *Multibody Syst. Dyn.*, vol. 43, no. 3, pp. 209–227, Jul. 2018.
[4] S. Molla-Alizadeh-Zavardehi, M. Hajiaghaei-Keshteli, and R. Tavakkoli-Moghaddam, "Solving a capacitated fixed-charge transportation problem by artificial immune and genetic algorithms with a Prüfer number representation," *Expert Syst. Appl.*, vol. 38, no. 8, pp. 10462–10474, Aug. 2011.
[5] R. He, C. Ma, C. Ma, W. Zhang, and Q. Xiao, "Optimisation algorithm for logistics distribution route based on Prufer codes," *Int. J. Wireless Mobile Comput.*, vol. 9, no. 2, pp. 205–210, 2015.
[6] Z. Hashemi and F. G. Tari, "A Prufer-based genetic algorithm for allocation of the vehicles in a discounted transportation cost system," *Int. J. Syst. Sci., Oper. Logistics*, vol. 5, no. 1, pp. 1–15, Jan. 2018.
[7] S. M. A. Nayeem and M. Pal, "Diameter constrained fuzzy minimum spanning tree problem," *Int. J. Comput. Intell. Syst.*, vol. 6, no. 6, pp. 1040–1051, Dec. 2013.
[8] A. Algergawy, E. Schallehn, and G. Saake, "Improving XML schema matching performance using Prüfer sequences," *Data Knowl. Eng.*, vol. 68, no. 8, pp. 728–747, Aug. 2009.
[9] C. Greenhill, M. Isaev, M. Kwan, and B. D. McKay, "The average number of spanning trees in sparse graphs with given degrees," *Eur. J. Combinatorics*, vol. 63, pp. 6–25, Jun. 2017.
[10] D. Catanzaro and R. Pesenti, "Enumerating vertices of the balanced minimum evolution polytope," *Comput. Oper. Res.*, vol. 109, pp. 209–217, Sep. 2019.
[11] J. Xu, Q. Liu, and R. Wang, "A class of multi-objective supply chain networks optimal model under random fuzzy environment and its application to the industry of chinese liquor," *Inf. Sci.*, vol. 178, no. 8, pp. 2022–2043, Apr. 2008.
[12] S. W. Singaram, A. Gopal, and A. Ben-Shaul, "A Prüfer-sequence based algorithm for calculating the size of ideal randomly branched polymers," *J. Phys. Chem. B*, vol. 120, no. 26, pp. 6231–6237, Jul. 2016.
[13] H.-Y. Su, C.-C. Chen, Y.-L. Li, A.-C. Tu, C.-J. Wu, and C.-M. Huang, "A novel fast layout encoding method for exact multilayer pattern matching with Prüfer encoding," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 1, pp. 95–108, Jan. 2015.
[14] T. Paulden and D. K. Smith, "Developing new locality results for the Prüfer code using a remarkable linear-time decoding algorithm," *Electron. J. Combinatorics*, vol. 14, no. 1, p. R55, Aug. 2007.
[15] H.-C. Chen and Y.-L. Wang, "An efficient algorithm for generating Prüfer codes from labelled trees," *Theory Comput. Syst.*, vol. 33, no. 1, pp. 97–105, Jan. 2000.
[16] X. Wang, L. Wang, and Y. Wu, "An optimal algorithm for Prufer codes," *J. Softw. Eng. Appl.*, vol. 2, no. 2, pp. 111–115, 2009.
[17] S. Caminiti, I. Finocchi, and R. Petreschi, "On coding labeled trees," *Theor. Comput. Sci.*, vol. 382, no. 2, pp. 97–108, Aug. 2007.
[18] J. Wang and K. M. Yan, "Linear algorithm of Prufer codec based on array," *J. Xi'an Shiyou Univ. (Natural Sci. Ed.)*, vol. 28, pp. 102–105, 2013.
[19] X. D. Wang and Y. J. Wu, "Optimal algorithm for coding and decoding Prufer codes," *J. Chin. Comput. Syst.*, vol. 29, no. 4, pp. 687–690, 2008.
[20] R. R. Liu, J. E. Chen, and S. Q. Chen, "Improvement of Warshall algorithm based on transitive closure," *Comput. Eng.*, pp. 38–39, 2005.
[21] J. He, J. Chen, G. Huang, J. Cao, Z. Zhang, H. Zheng, P. Zhang, R. Zarei, F. Sansoto, R. Wang, and Y. Ji, "A polynomial-time algorithm for simple undirected graph isomorphism," *Concurrency Comput., Pract. Exper.*, p. e5484, Aug. 2019.

LIN YANG received the B.Sc. degree in electronic engineering from the National University of Defense Technology, Hefei, China, where he is currently pursuing the M.Sc. degree in electromagnetic countermeasure. His research interests include cyberspace security, network security situational awareness, and artificial intelligence.

YONGJIE WANG received the M.Sc. and Ph.D. degrees from the National University of Defense Technology, Changsha, China. He is currently an Associate Professor of electronic engineering with the National University of Defense Technology, Hefei, China. His research interests include cyberspace security, risk assessment, and information system modeling and simulation.