Simplifying Calculation of Graph Similarity Through Matrices

Xu Wang, Jihong Ouyang, Guifen Chen

Simplifying Calculation of Graph Similarity Through Matrices

Xu Wang[1,2,a], Jihong Ouyang[1,2,b,*], Guifen Chen[3,c]

[1]College of Computer Science and Technology, Jilin University, Changchun130012, China;
[2]Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun130012, China;
[3]College of Information Technology, Jilin Agricultural University, Changchun 130118, China
[a]xuwang10@mails.jlu.edu.cn, [b]ouyj@jlu.edu.cn, [c]guifchen@163.com

Abstract. A method to simplify the calculation in the process of measuring graph similarity is proposed, where lots of redundant operations are avoided in order to quickly obtain the initial tickets matrix. In this proposal, the element value of the initial tickets matrix is assigned to 1 when it is positive in corresponding position of the paths matrix at the first time. The proposed method calculates the initial tickets matrix value based on the positive value in the paths matrix in a forward and backward way. An example is provided to illustrate that the method is feasible and effective.

Keywords: graph similarity measure, paths and tickets, adjacency matrix, forward and backward, simplifying calculation

1 Introduction

The complex objects with structural properties can be naturally established as graphs in chemistry, bioinformatics, data mining, social network, image processing and other fields [1-6]. For example, the science citation can be represented as a graph with the literatures as vertices and the indexes as edges. As the large amount of graph data is increasing, how to characterize the difference of the graph data becomes an important problem [7]. The graph similarity method that efficiently solves the problem can be applied to classify the graphical data.

In order to measure the graph similarity, we usually calculate the number of the common paths of the graph, such as random walk graph kernel (abbreviated as RWGK) [8][9], the shortest path graph kernel (abbreviated as SPGK) [10], and the function of the common paths or tickets [4], etc. The function of the common paths or tickets, in the spirit of the neighborhood counting and all common subsequences in measuring the sequence similarity [11-13], measures graph similarity by calculating all common paths or tickets matrices between two graphs [14-16]. Compared with RWGK and SPGK, this method considers more information of vertices and edges in graph. This method is remarkable in accuracy and running time, in addition, it is not restrictive and can be applied in larger graph dataset. Also, it avoids the tottering phenomenon [17]. In the process of calculating the tickets matrices, the initial tickets matrix is raised from all paths matrix, which is calculated by Floyd-Warshall

Algorithm [18-20]. When the element value in the paths matrix is positive, by using the method in [4], the value in the initial tickets matrix is always 1 in the corresponding position, in spite of the multiple calculations of matrices multiplying and addition. However, this method still continues calculating the element value of the paths matrix, and it generates lots of redundant calculations.

For this problem, in this paper, we propose a simplified method to calculate the initial tickets matrix. When the element value in the paths matrix of graph is positive at the first time, this method initializes the value as 1 in the corresponding position in the initial tickets matrix, instead of continuing calculating in [4], and set the value of the initial tickets matrix based on the positive value of the paths matrix in a forward and backward way. This method will avoid lots of redundant calculations, and can be applied in a large-scale complex graph data. An example is provided to illustrate its feasibility and efficiency.

This paper is organized as follows: Section 2 describes the related concepts and notations. Section 3 proposes a simplified calculation method to obtain the initial tickets matrix. Section 4 presents the analyses process of the proposed method. The paper is concluded with a summary and an outlook for future work in Section 5.

2 Preliminaries

A graph is usually represented as $G = (V, E)$, where the vertices set is $V = \{v_1, \ldots, v_n\}$ and the edges set is $E = \{(v_1, v_2), \ldots, (v_i, v_j)\}$. If vertex v_i is up to the vertex v_j, then there exists a path between v_i and v_j. If the length of two paths is equal, and the vertices and edges are the same, then we call the two paths as the common paths. The more these common paths in two graphs, the more similar the two graphs are [4]. Two graphs are perfectly similar when they share all paths, while two graphs are perfectly dissimilar when they share no paths. A graph can be denoted by its adjacency matrix A as the following equation:

$$a(i, j) = \begin{cases} 1 & if \ (v_i, v_j) \in E \\ 0 & otherwise \end{cases} \quad (1)$$

$a(i, j) = 1$ denotes there exists a distinct 1-long path from v_i to v_j, $\Sigma_{ij} a(i, j)$ denotes the number of all distinct 1-long paths in graph.

For three adjacency matrices A, B, C with n rows and n columns, if $C = A \times B$, then $c(i, j)$ is the element of C in i-th row and j-th column, and it can be calculated as $c(i, j) = \Sigma_k a(i, k) \times b(k, j)$, where $1 \le i \le n$, $1 \le j \le n$, $1 \le k \le n$. Obviously, it can be seen that $a^2(i, j)$ denotes a 2-long path from i to j passing the vertex k, which is calculated by the matrices multiplying of A^1 and A^1, $a^2(i, j) = \Sigma_k a^1(i, k) a^1(k, j)$. In other words, a 2-long path from i to j consists of a 1-long path from i to some k and a 1-long path pass vertex from that k to j. A 2-long path exists if and only if both $a(i, k)$ and $a(k, j)$ are positive, i.e. $a(i, k) > 0$, and $a(k, j) > 0$. The element in the matrix A^2($A^2 = A^1 \times A^1$) is the number of the 2-long paths in graph, so all possible 2-long paths from i to j is $\Sigma_{ij} a^2(i, j)$. In general, $a^k(i, j)$ denotes the number of distinct k-long paths from v_i to v_j, and $\Sigma_{ij} a^k(i, j)$ denotes the total number of k-long paths in graph, where $1 \le k \le n$. For example, $A^n = A^1 \times A^{n-1}$, the element in A^n is the number of the

possible *n*-long paths in graph. If $C = A + B$, we know $c(i, j) = a(i, j) + b(i, j)$, then the number of the distinct paths from *i* to *j* can be calculated as follows:

$$\begin{aligned} P^1 &= A^1 \\ P^2 &= P^1 + A^2 \\ &\text{M} \\ P^n &= P^{n-1} + A^n \\ P^n &= A^1 + A^2 + \text{L} + A^n,\ A^{n+1} = (0) \end{aligned} \tag{2}$$

$P^n(i, j) \geq 0$ denotes the number of the distinct paths from *i* to *j* that are at most *n*-long. $\Sigma_{ij} p^n(i, j)$ denotes the total number of paths in graph.

The measurement of the paths is usually restrictive, so we need to measure the tickets instead of measuring the paths. A ticket is a contracted path by deleting any number of nodes from a path, which is obtained from graph by deleting and contracting the isolated edges [4][21]. If the length of a path between v_i and v_j is more than 1, then there exists a 1-long ticket $v_i v_j$. The all paths matrix is a 1-long tickets matrix that can be calculated as follows:

$$t^1(i, j) = \begin{cases} 1 & if \quad p^n(i, j) \geq 1 \\ 0 & otherwise \end{cases} \tag{3}$$

By Eq. (3), there is $T^1 = P^n$. We call the matrix T^1 as the initial tickets matrix. The method of calculating all common tickets is similar to the method of calculating all the common paths. The more these common tickets, the more similar the graphs are [4].

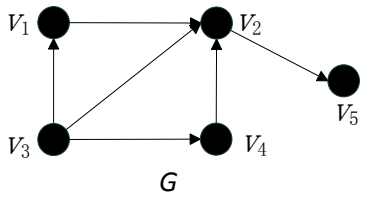
In the paper, the graph is directed and acyclic, and the length of the path we consider is finite. If a graph is a cycle graph, the length of the path with the cycle is infinite, so it cannot be used to measure all common paths or tickets.

3 A Simplified Calculation Method to Calculate Tickets Matrix

This section proposes a simplified calculation method of measuring graph similarity in order to quickly obtain the initial tickets matrix. In the process of calculating the initial tickets matrix, when the element value of the paths matrix is positive at the first time, this method sets the value as 1 in the corresponding position of the initial tickets matrix, and does not calculate the value again. It avoids the redundant calculations of matrices multiplying and addition.

In order to obtain the initial tickets matrix, based on Eq. (2) and Floyd-Warshall Algorithm in [19], we need to calculate all paths matrix P^n. When P^n is known, we can obtain the initial tickets matrix T^1 by using Eq. (3). Obviously, we note that the element value in the initial tickets matrix is always 1 when the value is positive in the corresponding position of the paths matrix. If the value in the paths matrix is positive at the first time, in spite of lots of calculations of matrices multiplying and addition in this position, the value is always positive. So we can conclude that if the value in the paths matrix is positive at the first time, the value in the initial tickets matrix must be equal to 1. It does not need the calculation in the corresponding position of the paths matrix again. The aim of the method in the paper is to reduce the redundant calculations when the element value in the paths matrix is positive.

An adjacency matrix of the graph G is shown in the Fig. 1, where $a(i, j) = 1$ denotes that there is a distinct 1-long path from v_i to v_j. $\Sigma_{ij}a(i, j)$ denotes the number of all distinct 1-long paths in graph G.



$$A^1=\begin{bmatrix}0&1&0&0&0\\0&0&0&0&1\\1&1&0&1&0\\0&1&0&0&0\\0&0&0&0&0\end{bmatrix}$$

Fig. 1. The directed and acyclic graph G and its adjacency matrix A^1.

3.1 Simplified Calculation When 1-long Paths is Fewer

When the number of 1-long paths is fewer than n^2 in the adjacency matrix of graph, where n^2 denotes the element number of the adjacency matrix, based on the adjacency matrix A^1, we calculate the initial tickets matrix T^1 as follows: $t^1(i, j) = a^1(i, j)$, where $1 \le i \le n$, $1 \le j \le n$. For $a^1(i, j) = 1$, it represents that there exists a 1-long path between v_i and v_j. In other words, there exists a ticket $t^1(i, j) = 1$, and we don't calculate $t^1(i, j)$ again. It reduces the redundant calculations in i-th row and j-th column of T^1. When the element value in T^1 is 0, we can calculate the value in i backward and j forward step.

$$t^1(k, j) = \begin{cases} 1 & if\ a^1(k,i) = 1\ and\ a^1(k, j) = 0 \\ 0 & otherwise \end{cases} \tag{4}$$

If $a^1(k, i) = 1$, where $1 \le k \le n$, $k \ne i$ and $k \ne j$, it shows that v_kv_i is a path in the graph, i is the backward vertex pointed by k. It implies that there exists a 1-long ticket between v_k and v_j, then $t^1(k, j) = 1$.

$$t^1(i, k) = \begin{cases} 1 & if\ a^1(j, k) = 1\ and\ a^1(i, k) = 0 \\ 0 & otherwise \end{cases} \tag{5}$$

If $a^1(j, k) = 1$, where $1 \le k \le n$, $k \ne i$ and $k \ne j$, it shows that v_jv_k is a path in the graph, j is the forward vertex pointing to k. It implies that there exists a 1-long ticket between v_i and v_k, then $t^1(i, k) = 1$.

By using Eq. (4) and Eq. (5), we can calculate element values of the initial tickets matrix T^1 in accordance with all paths matrix P^n. Because the number of $\{1 \le i \le n, 1 \le j \le n \mid a^1(i, j) = 1\}$ is small, the calculation of $t^1(i, j)$ is not complicated, which means this method reduces redundant calculations of matrices multiplying and addition in comparison with Floyd-Warshall Algorithm.

3.2 Simplified Calculation When 1-long Paths is More

When the number of 1-long paths is closer to n^2 in the adjacency matrix of graph, where n^2 denotes the element number of the adjacency matrix, based on the adjacency matrix A^1, we also calculate the initial tickets matrix T^1 as follows: $t^1(i, j) = a^1(i, j)$,

where $1 \le i \le n$, $1 \le j \le n$. When the element value in T^1 is 1, we don't calculate the value again. It reduces the redundant calculations in the position of the value. For $a^1(i, j) = 0$, it indicates that there doesn't exist a 1-long path between v_i and v_j, we can calculate the $t^1(i, j)$ value of the initial tickets matrix T^1 in i forward and j backward step.

$$t^1(i, j) = \begin{cases} a^1(k, j) & if\ a^1(i, k) = 1 \\ 0 & otherwise \end{cases} \quad (6)$$

If $a^1(i, k) = 1$, where $1 \le k \le n$, $k \ne i$ and $k \ne j$, it shows that $v_i v_k$ is a path in the graph, i is the forward vertex pointing to k. The problem that whether there exists a path between v_i and v_j is changed to the problem that whether there exists a n-long path between v_k and v_j, where $n \ge 1$, then $t^1(i, j)$ recursively is equivalent to $a^1(k, j)$.

$$t^1(i, j) = \begin{cases} a^1(i, k) & if\ a^1(k, j) = 1 \\ 0 & otherwise \end{cases} \quad (7)$$

If $a^1(k, j) = 1$, where $1 \le k \le n$, $k \ne i$ and $k \ne j$, it shows that $v_k v_j$ is a path in the graph, j is the backward vertex pointed by k. The problem that whether there exists a path between v_i and v_j is changed to the problem that whether there exists a n-long path between v_i and v_k, where $n \ge 1$, then $t^1(i, j)$ recursively is equivalent to $a^1(i, k)$.

By using Eq. (6) and Eq. (7), we can calculate element values of the initial tickets matrix T^1 in accordance with all paths matrix P^n. Because the number of $\{1 \le i \le n, 1 \le j \le n \mid a^1(i, j) = 1\}$ in the adjacency matrix is large, then the number of $\{1 \le i \le n, 1 \le j \le n \mid a^1(i, j) = 0\}$ is small, so the calculation of $t^1(i, j)$ also is not complicated. In other words, this method also reduces redundant calculations of matrices multiplying and addition comparing to Floyd-Warshall Algorithm.

In the spirit of measuring the sequence similarity by all common subsequences, a graph similarity measure method is proposed by calculating all common tickets. By means of the measuring sequence accuracy, the accuracy is more precise by calculating all common tickets than calculating all common paths. In order to calculate all common tickets in the process of measuring the graph similarity, how to quickly obtain the initial tickets matrix is a key problem. The method mentioned above, which avoids lots of redundant calculations of matrices multiplying and addition in the process of calculating the initial tickets matrix, can effectively solve the problem.

4 Method Analyses

This section gives two algorithms to simplify calculation of the initial tickets matrix, and illustrates its feasibility by providing an example.

4.1 The Algorithm Analyses

Calculating all paths matrix P^n directly obtains the complexity is $O(n^4)$, but based on the Floyd-Warshall algorithm, an algorithm has been designed which has a complexity of $O(n^3)$ [19].

Floyd-Warshall Algorithm:

Let $A^1 = a(i, j)$ be an $n \times n$ adjacency matrix of graph.

1. $P \leftarrow A^1$
2. for $k = 1$ to n do
3. for $i = 1$ to n do
4. for $j = 1$ to n do
5. $p(i, j) = p(i, j) + p(i, k) \times p(k, j)$
6. end for
7. end for
8. end for
9. return P

If A^1 is the adjacency matrix of a graph G, then P gives the number of all paths in graph G. If A^1 is the common adjacency matrix of two or more graphs, then P gives the number of all common paths. The return matrix P is used to calculate the initial tickets matrix.

As presented in Eq. (4) and Eq. (5), when the number of the 1-long paths is fewer than n^2, we present the idea of simplifying calculation of the initial tickets matrix as follows:

Algorithm 1: The simplified calculation of the initial tickets matrix T^1

Let $A = a(i, j)$ be an $n \times n$ adjacency matrix of graph.

1. $T^1 \leftarrow A^1$
2. for $i = 1$ to n do
3. for $j = 1$ to n do
4. if $t^1(i, j) > 0$
5. $t^1(i, j) = 1$; // When $t^1(i, j) > 0$, initialize the tickets matrix T^1
6. $Q \leftarrow t^1(i, j)$; //When $t^1(i, j) = 1$, put $t^1(i, j)$ in the queue Q, i and j are known
7. end
8. end for
9. end for
10. while $|Q| > 0$ //Because of $|\{a^1(i, j) = 1\}|$ is small, the implement times are fewer
11. $t^1(i, j) = q \leftarrow Q$ // Take out an element from queue Q
12. for ($k = 1$, $k \neq i$ and $k \neq j$) to n do
13. if $t^1(j, k) = 1$ and $t^1(i, k) \notin q$ // There is a path from j to k in the matrix A and $t^1(i, k)$ is not in queue Q
14. $T^1 \leftarrow t^1(i, k) = 1$; // $v_i v_k$ is a ticket, initialize the matrix T^1: $t^1(i, k) = 1$
15. end
16. if $t^1(k, i) = 1$ and $t^1(k, j) \notin q$ // There is a path from k to i in the matrix A and $t^1(k, j)$ is not in queue Q
17. $T^1 \leftarrow t^1(k, j) = 1$; // $v_k v_j$ is a ticket, initialize the matrix T^1: $t^1(k, j) = 1$
18. end
19. end for
20. $Q = Q - \{q\}$ // Remove q from Q

21. return T^1

As shown in Eq. (6) and Eq. (7), when the number of the 1-long paths is closer to n^2, we present the idea of simplifying calculation of the initial tickets matrix as follows:

Algorithm 2: The simplified calculation of the initial tickets matrix T^1

Let $A^1 = a^1(i, j)$ be an $n \times n$ adjacency matrix of graph.

1. $T^1 \leftarrow A^1$
2. for $i = 1$ to n do
3. for $j = 1$ to n do
4. if $t^1(i, j) > 0$
5. $t^1(i, j) = 1$; // When $t^1(i, j) > 0$, initialize the matrix T^1
6. else
7. $Q \leftarrow t^1(i, j)$ // When $t^1(i, j) = 0$, put $t^1(i, j)$ in the queue Q, i and j are known.
8. end
9. end for
10. end for
11. while $|Q| > 0$ //Because of $|\{a^1(i, j) = 0\}|$ is small, the implement times are fewer
12. $t^1(i, j) = q \leftarrow Q$ // Take out an element from queue Q
13. $m = j$; // Assign the subscript j to m
14. $h = i$; // Assign the subscript i to h
15. for ($k = 1$, $k \neq i$ and $k \neq j$) to n do
16. if $t^1(k, j) = 1$ // Find out k pointing to j in the matrix A
17. $j = k$; // j points to the forward vertex k
18. if $t^1(i, j) = 1$ //It shows that there is a n-long path from i to j, where $n \geq 1$, so there is a ticket from i to m, in other words, $q = 1$
19. $t^1(i, m) = 1$;
20. $T^1 \leftarrow t^1(i, m)$; // Initialize the matrix T^1: $q = t^1(i, m) = 1$
21. end
22. end
23. if $t^1(i, k) = 1$ // Find out k pointed by i in the matrix A
24. $i = k$; // i points to the backward vertex k
25. if $t^1(i, j) = 1$ //It shows that there is a n-long path from i to j, where $n \geq 1$, so there is a ticket from h to j, in other words, $q = 1$
26. $t^1(h, j) = 1$;
27. $T^1 \leftarrow t^1(h, j)$; // Initialize the matrix T^1: $q = t^1(h, j) = 1$
28. end
29. end
30. end for
31. $Q = Q - \{q\}$ // Remove q from Q
32. return T^1

The return matrix T^1 is the initial tickets matrix, in which the value of element is 1 or 0. The matrix T^1 is used to calculate the number of all common tickets in two or more graphs for measuring graphs similarity. The queue Q is used to store $t^1(i, j)$. It can be seen from line 10 in Algorithm 1 and line 11 in Algorithm 2, the smaller the length of the queue, the more quickly two algorithms implement. So, Algorithm 1 is suitable to the adjacency matrix that the number of the 1-long paths is fewer than n^2, and Algorithm 2 is suitable to the adjacency matrix that the number of the 1-long

paths is closer to n^2. Compared with line 5 in Floyd-Warshall Algorithm, Algorithm 1 and Algorithm 2 can effectively solve the redundant calculations in the loop of line 12 and line 15, respectively. The two algorithms usually could be easily implemented and it can effectively handle the big graphs.

4.2 Example

For the directed and acyclic graph G as shown in Fig. 1, we calculate the initial tickets matrix of G by using Eq. (2) and Eq. (3). Based on $A^k = A^1 \times A^{k-1}$, $A^k = (0)$ for k > 3, we can obtain the 2-long paths matrix A^2 and the 3-long paths matrix A^3 as shown in Fig. 2.

$$A^2=A^1\times A^1=\begin{bmatrix}0&0&0&0&1\\0&0&0&0&0\\0&2&0&0&1\\0&0&0&0&1\\0&0&0&0&0\end{bmatrix} \qquad A^3=A^1\times A^2=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&2\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}$$

Fig. 2. The 2-long paths matrix A^2 and 3-long paths matrix A^3.

By using Eq. (2) and Eq. (3), we can obtain all paths matrix P^n and the initial tickets matrix T^1 as shown in Fig. 3, and its computational complexity is $O(n^4)$.

$$P^n=A^1+A^2+A^3=\begin{bmatrix}0&1&0&0&1\\0&0&0&0&1\\1&3&0&1&3\\0&1&0&0&1\\0&0&0&0&0\end{bmatrix} \qquad T^1=\begin{bmatrix}0&1&0&0&1\\0&0&0&0&1\\1&1&0&1&1\\0&1&0&0&1\\0&0&0&0&0\end{bmatrix}$$

Fig. 3. All paths matrix P^n and the initial tickets matrix T^1.

We also calculate the initial tickets matrix T^1 by using Floyd-Warshall Algorithm. By using line 2 to line 8 in Floyd-Warshall Algorithm, we can calculate the number of all paths $p(i, j)$, where $1 \le i \le n$, $1 \le j \le n$. After calculating all paths matrix P, we obtain the initial tickets matrix T^1 by using Eq. (3), and its computational complexity is $O(n^3)$.

For the adjacency matrix A^1 as shown in Fig. 1, the number of 1-long paths is 6, it is fewer than 25(n = 5, n^2 = 25), so we apply Algorithm 1 to calculate the initial tickets matrix T^1 as follows:

Step 1: In accordance with line 2 to line 9 in Algorithm 1, we set the initial tickets matrix $T^1 = A^1$ as shown in Fig. 4(a) and the queue Q:{$t^1(v_1, v_2) = 1$, $t^1(v_2, v_5) = 1$, $t^1(v_3, v_1) = 1$, $t^1(v_3, v_2) = 1$, $t^1(v_3, v_4,) = 1$, $t^1(v_4, v_2) = 1$}.

Step 2: For the $q \in Q$, when $q = t^1(v_1, v_2) = 1$, there exists $t^1(v_2, v_5) = 1$ and $t^1(v_3, v_1) = 1$, it shows that v_1, v_2, v_5 and v_3, v_1, v_5 are 2-long paths. In other words, v_1v_5 and v_3v_5

are 1-long tickets. By using the line 12 to line 19 in Algorithm 1, we can calculate new tickets $t^1(v_1, v_5) = 1$ and $t^1(v_3, v_5) = 1$ as shown in Fig. 4(b).

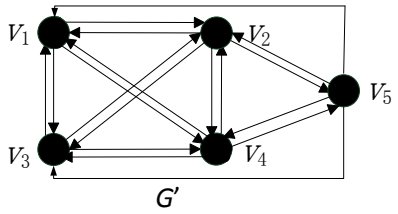
Similarly, we can calculate the final T^1 as presented in Fig. 4(c), the values identified by the box are the new tickets. When $t^1(v_i, v_j) = 1$, where $1 \le i \le n$, $1 \le j \le n$, Algorithm 1 doesn't need to repeatedly calculate $t^1(v_i, v_j)$. It reduces redundant calculations of matrices multiplying and addition comparing to Floyd-Warshall Algorithm.

$$T^1 = A^1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad T^1 = \begin{bmatrix} 0 & 1 & 0 & 0 & \boxed{1} \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & \boxed{1} \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{when } q = t^1(v_1, v_2). \quad T^1 = \begin{bmatrix} 0 & 1 & 0 & 0 & \boxed{1} \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & \boxed{1} \\ 0 & 1 & 0 & 0 & \boxed{1} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a) (b) (c)

Fig. 4. (a) Assigned T^1 using the adjacency matrix A^1; (b) when $q = t^1(v_1, v_2)$, we calculate T^1 using Algorithm 1; (c) we obtain the final T^1 based on Algorithm 1. The values identified by the box are the new tickets.

For a directed and acyclic graph G' and its adjacency matrix A'^1 as shown in Fig. 5, the number of the 1-long paths is 18, it is closer to 25($n = 5$, $n^2 = 25$), so we apply Algorithm 2 to calculate the initial tickets matrix T'^1 as follows:



$$A'^1 = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Fig. 5. The directed and acyclic graph G' and its adjacency matrix A'^1.

Step 1: In accordance with line 2 to line 10 in Algorithm 2, we set the initial tickets matrix $T'^1 = A'^1$ as shown in Fig. 6(a) and the queue Q: $\{t'^1(v_1, v_5) = 0, t'^1(v_3, v_5) = 0\}$. In this paper, the path doesn't contain a cycle. In other words, we don't consider the cycle $v_i, \ldots, v_k, \ldots, v_i$, where $1 \le i \le n$, $1 \le k \le n$. We set $t'^1(v_i, v_i)$ as 0 in the initial tickets matrix T'^1. So, in line 7 of Algorithm 2, we don't need to put $t'^1(v_i, v_i)$ in the queue Q.

Step 2: For the $q \in Q$, when $q = t'^1(v_1, v_5) = 0$, there exists $t'^1(v_1, v_2) = 1$ and $t'^1(v_2, v_5) = 1$, it shows that there is a 2-long path from v_1 to v_5. In other words, v_1v_5 is a 1-long ticket. By using the line 15 to line 30 in Algorithm 2, we can calculate new ticket $t'^1(v_1, v_5) = 1$ as shown in Fig. 6(b).

Similarly, we can calculate the final T'^1 as shown in Fig. 6(c), the values identified by the box are the new tickets. When $t'^1(v_i, v_j) = 1$, where $1 \le i \le n$, $1 \le j \le n$, Algorithm 2 doesn't need to repeatedly calculate $t'^1(v_i, v_j)$. It reduces redundant calculations of matrices multiplying and addition comparing to Floyd-Warshall Algorithm.

$$T'^1 = A'^1 = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad T'^1 = \begin{bmatrix} 0 & 1 & 1 & 1 & \boxed{1} \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad when\ q = t'^1(v_1, v_5).\ T'^1 = \begin{bmatrix} 0 & 1 & 1 & 1 & \boxed{1} \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & \boxed{1} \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

(a) (b) (c)

Fig. 6. (a) Assigned T'^1 using the adjacency matrix A'^1; (b) when $q = t'^1(v_1, v_5)$, we calculate T'^1 using Algorithm 2; (c) we obtain the final T'^1 based on Algorithm 2. The values identified by the box are the new tickets.

Let l denote the length of the queue Q, l is usually a constant in the best case. In Algorithm 1, the computational complexity is $O(n^2)$ in the loop of line 2 and line 3 and $O(lk)$ in the loop line 10 and line 12, where $1 \le k \le n$, $k \ne i$ and $k \ne j$. Similarly, in Algorithm 2, the computational complexity is $O(n^2)$ in the loop of line 2 and line 3 and $O(lk)$ in the loop of line 11 and line 15, where $1 \le k \le n$, $k \ne i$ and $k \ne j$. So the computational complexity of Algorithm 1 and Algorithm 2 is $O(n^2 + lk)$ and less than $O(n^3)$, approximately $O(n^2)$ in the best case. However, the computational complexity of using Eq. (2) and Eq. (3) is $O(n^4)$, and Floyd-Warshall Algorithm is $O(n^3)$. In Table 1, we list the complexity of calculating the initial tickets matrix algorithms. Obviously, Algorithm 1 and Algorithm 2 are more effective.

Table 1. The computational complexity of calculating the initial tickets matrix algorithms

Calculating algorithms	Computational complexity
Eq. (2) and Eq. (3)	$O(n^4)$
Floyd-Warshall Algorithm	$O(n^3)$
Algorithm 1	$O(n^2 + lk)$)
Algorithm 2	$O(n^2 + lk)$)

5 Conclusions

This paper proposes a simplified calculation method in the process of measuring graph similarity. This method sets the element value of the initial tickets matrix as 1 when the element value of the paths matrix is positive at the first time, and reduces lots of redundant calculations of matrices multiplying and addition comparing to Floyd-Warshall Algorithm. Depending on the number of 1-long paths in the adjacency matrix, this paper presents two algorithms to calculate the initial tickets matrix. The two algorithms calculate the element value of the initial tickets matrix based on the positive value of the paths matrix in a forward and backward way. By the algorithm analyses and the given example, these algorithms are proved to be feasible and effective. When we calculate the initial tickets matrix of the complex graph with large number of 1-long paths in the adjacency matrix, Algorithm 2 is more applicable, because the length of the stored queue is small.

In the near future, we will compare the method in this paper with graph kernel, random walk graph kernel, and the kernel function of all common paths or tickets in classification accuracy and running time through experiments. By means of easy calculation of this method, we will apply it to the undirected graph or the complex graph datasets.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Nos. 61133011, 61170092, 60973088, 60873149).

References

[1] Bapat Ravindra B. Graphs and Matrices [M]. New York: Springer Press, 2011.
[2] Hattori Masahiro, Okuno Yasushi, Goto Susumu et al. Development of a chemical structure comparison method for integrated analysis of chemical and genomic information in the metabolic pathways [J]. Journal of the American Chemical Society, 2003, 125(39): 11853-11865.
[3] Sperschneider Volker. Bioinformatics, Problem Solving Paradigms [M]. Berlin: Springer Press, 2008.
[4] H.Elzinga Cees, Wang Hui. Kernels for acyclic digraphs [J]. Pattern Recognition Letters, 2012, 33(16): 2239-2244.
[5] Kang U, Tong Hanghang, Sun Jimeng. Fast random walk graph kernel [C]. Proceedings of the Twelfth SIAM International Conference on Data Mining, 2012: 828-838.
[6] Kheradmand Amin, Milanfar Peyman. A General Framework for Regularized, Similarity-Based Image Restoration [J]. IEEE Transactions on Image Processing, 2014, 23(12): 5136-5151.
[7] Yuan Ye, Wang Guoren, Chen Lei et al. Graph similarity search on large uncertain graph databases [J]. The International Journal on Very Large Data Bases, 2015, 24(2): 271-296.
[8] Shervashidze Nino, M. Borgwardt Karsten [C]. Fast subtree kernels on graphs. The annual Conference on Neural Information Processing Systems, 2009: 1660-1668.
[9] Vishwanathan S.V.N., N.Schraudolph Nicol, Kondor Risi et al. Graph Kernels [J]. Journal of Machine Learning Research, 2010, 11: 1201-1242.
[10] M.Borgwardt Karsten, Kriegel Hans-Peter. Shortest-Path Kernels on Graphs [C]. IEEE International Conference on Data Mining, 2005: 74-81.
[11] Wang Hui. Nearest Neighbors by Neighborhood Counting [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2006, 28(6): 942-953.
[12] Wang Hui. All Common Subsequences [C]. Proceeding 20th International Joint Conference on Artificial Intelligence, 2007: 635-640.
[13] Wang Hui, Murtagh Fionn. A Study of the Neighborhood Counting Similarity [J]. IEEE Transactions on Knowledge and Data Engineering, 2008, 20(4): 449-461.
[14] He Jun, Liu Hongyan, Yu Jeffrey Xu et al. Assessing single-pair similarity over graphs by aggregating first-meeting probabilities [J]. Information Systems, 2014, 42: 107- 122.
[15] Kramer Michael, Dutkowski Janusz, Yu Michael et al. Inferring gene ontologies from pairwise similarity data [J]. Bioinformatics, 2014, 30(12): 34-42.

[16] Rodriguez Andrew, Kim Byunghoon, Lee Jaemin et al. Graph kernel based measure for evaluating the influence of patents in a patent citation network [J]. Expert Systems with Applications, 2015, 42(3): 1479-1486.
[17] Mahé Pierre, Ueda Nobuhisa, Akutsu Tatsuya et al. Extensions of marginalized graph kernels [C]. Proceedings of the 21st International Conference on Machine Learning, 2004:552-559.
[18] Robert W Floyd. Algorithm 97: Shortest Path [J]. Communications of the ACM, 1962, 5(6): 345.
[19] Kása Zoltán. On scattered subword complexity [J]. Acta Univ. Sapientiae, Informatica, 2011, 3(1): 127–136.
[20] Wang, Ilin. An algebraic decomposed algorithm for all pairs shortest paths [J]. Pacific journal of optimization, 2014, 10(3): 561-576.
[21] Diestel Reinhard. Graph Theory [M]. Third Edition. New York: Springer Press, 2005.