Research Article

A novel graph matrix representation: sequence of neighbourhood matrices with an application

Sivakumar Karunakaran[1] · **Lavanya Selvaganesh**[2]

Received: 31 October 2019 / Accepted: 30 March 2020 / Published online: 22 April 2020

Abstract
In the study of network optimization, finding the shortest path minimizing time/distance/cost from a source node to a destination node is one of the fundamental problems. Our focus here is to find the shortest path between any pair of nodes in a given undirected unweighted simple graph with the help of the sequence of powers of neighbourhood matrices. The authors recently introduced the concept of neighbourhood matrix as a novel representation of graphs using the neighbourhood sets of the vertices. In this article, an extension of the above work is presented by introducing a sequence of matrices, referred to as the sequence of powers of $\mathcal{NM}(G)$. It is denoted it by $\mathcal{NM}^{\{l\}}(G) = [\eta_{ij}^{\{l\}}]$, $1 \leq l \leq k(G)$, where $k(G)$ is called the iteration number, $k(G) = \lceil \log_2 \text{diameter}(G) \rceil$. As this sequence of matrices captures the distance between the nodes profoundly, we further develop the technique and present several characterizations. Based on the theoretical results, we present an algorithm to find the shortest path between any pair of nodes in a given graph. The proposed algorithm and the claims therein are formally validated through simulations on synthetic data and the real network data from Facebook. The empirical results are quite promising with our algorithm having best running time among all the existing well-known shortest path algorithms for the considered graph classes.

Keywords Shortest path problem · Graph matrix · Neighbourhood matrix · Matrix multiplication · Complex network analysis

Mathematics Subject Classification 05C50 · 05C62 · 05C82 · 05C85

1 Introduction

With the advent of modern technology and electronic devices, the occurrence of large-scale networks is widespread and ubiquitous. Especially in the field of computer science, these networks are omnipresent in various forms such as communication networks and social networks. These networks require cooperation between a large number of individual components/parts. Major challenges in this field include predicting the behaviour of the large-scale network, and information propagation within the network. For every large-scale network that arises in daily life, their understanding and mathematical description are crucial and achieved with the aid of an intricate graph representation that encodes the interaction between the network's components.

One of the fundamental problems in the study of network optimization is the shortest path problem that deals with finding a path with minimum distance, time or cost from the source to the destination. Most of the

A preliminary version of this article is also available in the arXiv with the identifier 1903.05377.

✉ Lavanya Selvaganesh, lavanyas.mat@iitbhu.ac.in; Sivakumar Karunakaran, sivakumar_karunakaranm@srmuniv.edu.in | [1]SRM Research Institute, SRM Institute of Science and Technology, Kattankulathur, Chennai, Tamil Nadu 603203, India. [2]Department of Mathematical Sciences, Indian Institute of Technology (BHU) Varanasi, Varanasi, Uttar Pradesh 221005, India.

communication network problems, such as information propagation/exchange, were modelled as designing/finding multiple routes between nodes (if it exists) and finding shortest paths between nodes. Two classical approaches for the single-source shortest path problem are attributable to Dijkstra [8] and Bellman–Ford [2]. For more, see [7]. Since then, many researchers have been working on finding the shortest path by reducing the involved time complexity. For a survey of the shortest path algorithms, we refer the readers to [14, 18, 26, 30]. Further, in the literature, many variations and special cases of the shortest path problem were proposed. For instance, we refer to the recent works on massively parallel systems [5], dynamic algorithms for hypergraphs [12], random networks with uncertain arc lengths [13, 25] and in fuzzy graphs [9, 11]. There are numerous algorithms and techniques proposed by many researchers; see [1, 4, 10, 15, 19, 22–24] for probabilisitic approach and for large graphs, we refer to [20, 21], and [29] exploits the symmetry in graphs, for more related literature.

In this paper, our focus is in developing algorithm for the single-source shortest path (SSSP) problem over large-scale graphs in a much efficient manner using algebraic graph theory and theory of matrices. That is, given an undirected graph G of a network, design an efficient algorithm to find shortest paths between a pairs of nodes by using the matrix introduced in [16]. This novel graph matrix associated with a graph was called neighbourhood matrix $\mathcal{NM}(G)$ and is defined as,

$$\eta_{ij} = \begin{cases} -|N_G(i)|, & \text{if } i = j \\ |N_G(j) - N_G(i)|, & \text{if } (i,j) \in E(G) \\ -|N_G(i) \cap N_G(j)|, & \text{if } (i,j) \notin E(G) \end{cases},$$

where $N_G(v)$ denote the set of all neighbours of the vertex v in G. In the same paper, the authors also have discussed various properties and characterizations of $\mathcal{NM}(G)$.

We present an extension of the work mentioned above and introduce a scheme for associating a finite sequence of matrices to an undirected graph G. As an application of this scheme, we design an algorithm to compute the shortest path between any pair of vertices. We then discuss the running time and efficiency of the algorithm by supporting empirical results. The proposed algorithm and the claims therein are formally validated through simulations on synthetic data where sampling-based computations are performed in large scale and with large-sized graphs. The empirical results are quite promising, wherein the proposed algorithm has least running time among all the existing well-known shortest path algorithms such as Dijkstra, Bellman–Ford and breadth-first search (BFS). The $\mathcal{NM}$-ShortestPath algorithm can be used effectively in real-life situations involving communication networks where disruption in the network are not tolerated and computing the alternate routes in the network needs to be done within a micro-fraction of a second irrespective of the weights involved in them.

This paper is organized as follows: Rest of this section describes all the basic definitions, notations and also introduce the novel concept of sequence of powers of neighbourhood matrix. Section 2 discusses the methodology with the mathematical setting of the graph characterizations from the matrix entries and various results on the sequence of powers of neighbourhood matrix. Following this, we present the algorithm and the pseudocode (written in *MATLAB*) to find the shortest path between any two vertices in a graph and analyse the complexity of the proposed algorithms. In Sect. 3, we present numerical results supporting our theory with simulations and empirical results. Conclusions with future directions follow in Sect. 4. Lastly, for the sake of completion, proofs of all the results discussed in this paper are presented in "Appendix".

1.1 Neighbourhood matrix

In this subsection, we present all the required notations, definitions of the neighbourhood matrix $\mathcal{NM}(G)$. For all basic notations and definitions of graph theory, we follow the books by Bondy and Murty [3] and West [28]. All the graphs considered are undirected, unweighted and simple. Let $G(V(G), E(G))$ be a graph with vertex set $V(G)$ and edge set $E(G)$. For a vertex $v \in V(G)$, let $N_G(v)$ denote the set of all neighbours of v. The degree of a vertex v is given by $d_G(v)$ or $|N_G(v)|$. The diameter of the graph is denoted as *diameter*(G) and the shortest path distance between two vertices i and j in G is denoted by $d_G(i,j), i,j \in V(G)$. Let $A_G, D(G)$ and $C(G)$ denote the adjacency matrix, degree matrix and the Laplacian/admittance matrix of the graph G, respectively. For readers interested in the proofs of the results in this section, we refer to [16].

Definition 1 [16] Given a graph G, the neighbourhood matrix, denoted by $\mathcal{NM}(G) = (\eta_{ij})$ is defined as

$$\eta_{ij} = \begin{cases} -|N_G(i)|, & \text{if } i = j \\ |N_G(j) - N_G(i)|, & \text{if } (i,j) \in E(G) \\ -|N_G(i) \cap N_G(j)|, & \text{if } (i,j) \notin E(G) \end{cases}$$

The following lemma facilitates us with a way of computing the neighbourhood matrix.

Lemma 1 [16] *The $\mathcal{NM}(G)$ can also be obtained from the product of adjacency matrix and Laplacian matrix of a graph G.*

For the computation of the $\mathcal{NM}(G)$, we often resort to the above lemma. Several efficient algorithms have been proposed in the literature to compute product/multiplication of two matrices; see [6, 27]. We make use of the algorithm described in [6].

Another concept which we require before proceeding to the main result is the level decomposition of a graph from a source node, which is defined by the breadth-first search traversal technique.

Breadth-first search (BFS) is a graph traversal technique [3] where a (source) node and its neighbours are visited first and then the neighbours of neighbours. The algorithm returns not only a search tree rooted at the source node but also a function $l : V \to \mathbb{N}$, which records the level of each vertex in the tree, that is, the distance of each vertex from the source node. In simple terms, the BFS algorithm traverses level-wise from the source. First, it traverses level 1 nodes (direct neighbours of source node) and then level 2 nodes (neighbours of neighbours of the source node) and so on. We refer to such a level representation as to the level decomposition from the source node.

Lemma 2 [16] *Given a graph G and a vertex v, the entries of any row of $\mathcal{NM}(G)$ correspond to the subgraph with vertices from the first two levels of the level decomposition of the graph rooted at v with all the edges connecting the vertices lying in different levels.*

Remark 1 [16] The proof of the above lemma (given in "Appendix") also reveals interesting facts about the neighbourhood matrix which justifies its terminology and further helps us in proving the following theorem.

Theorem 1 [16] *If at least one row of $\mathcal{NM}(G)$ has no zero entries, then the graph G has diameter at most 4.*

Remark 2 [16] Note the converse of the above theorem need not hold. For example, consider the hypercube Q_3, whose diameter is 3 while every row of the $\mathcal{NM}(Q_3)$ matrix (see Fig. 1) contains a zero entry.

$$\mathcal{NM}(Q_3) = \begin{pmatrix} -3 & 3 & -2 & 0 & -2 & 3 & -2 & 3 \\ 3 & -3 & 3 & -2 & 3 & -2 & 0 & -2 \\ -2 & 3 & -3 & 3 & -2 & 0 & -2 & 3 \\ 0 & -2 & 3 & -3 & 3 & -2 & 3 & -2 \\ -2 & 3 & -2 & 3 & -3 & 3 & -2 & 0 \\ 3 & -2 & 0 & -2 & 3 & -3 & 3 & -2 \\ -2 & 0 & -2 & 3 & -2 & 3 & -3 & 3 \\ 3 & -2 & 3 & -2 & 0 & -2 & 3 & -3 \end{pmatrix}$$

Fig. 1 The neighbourhood matrix of the hypercube Q_3

Theorem 2 [16] *The matrix $\mathcal{NM}(G)$ has no zero entries if and only if the graph G has diameter at most 2.*

1.2 Sequence of neighbourhood matrices

Theorem 2 captures the essence of the vertices which are placed at a distance at most 2. We extend this idea and introduce in this article an iterative procedure to generate a sequence of graphs and consequently, a sequence of $\mathcal{NM}$ matrices associated with these graphs. Such a sequence of $\mathcal{NM}$ matrices has proven to be useful in reducing the computational time involved for finding various properties of a given graph.

In the next section, we show one such application of this process in finding the shortest path between any pair of vertices.

Definition 2 Let G be an undirected unweighted simple graph on n vertices. The sequence of neighbourhood matrices $< \mathcal{NM}^{\{1\}} >$ is recursively defined as follows: Let $G^{\{1\}} = G$ and $\mathcal{NM}^{\{1\}} = \mathcal{NM}(G^{\{1\}})$. For $l > 1$, let $G^{\{l\}}$ be the graph constructed from the graph $G^{\{l-1\}}$ as follows: $V(G^{\{l\}}) = V(G^{\{l-1\}})$ and $E(G^{\{l\}}) = E(G^{\{l-1\}}) \cup \{(i,j) : \eta_{ij}^{\{l-1\}} < 0, i \neq j\}$. Then, the matrix $\mathcal{NM}^{\{l\}}$ is defined to be $\mathcal{NM}(G^{\{l\}})$ and can be constructed from $G^{\{l\}}$ using Definition 1 or Lemma 1.

Remark 3 Note that the adjacency matrix of $A(G^{\{l\}})$ is given by:

$$A(G^{\{l\}}) = (a_{ij}^{\{l\}}) = \begin{cases} 1, & \text{if } \eta_{ij}^{\{l-1\}} \neq 0, i \neq j \\ 0, & \text{Otherwise} \end{cases}$$

Definition 3 Let k be the smallest possible integer, such that, $\mathcal{NM}^{\{k\}}$ has no zero entry or the number of nonzero entries of $\mathcal{NM}^{\{k\}}$ and $\mathcal{NM}^{\{k+1\}}$ are the same. The number $k(G) := k$ is called the iteration number of G.

Remark 4 When $G = K_n$, the complete graph, $\mathcal{NM}(K_n)$ has no zero entries. Hence for K_n, the iteration number is $k(K_n) = 1$. Further, for a graph G with diameter 2, $\mathcal{NM}(G)$ has no zero entries, hence iteration number $k(G) = 1$

Let us illustrate the above definitions with a randomly generated graph G on 12 vertices and 14 edges given in Fig. 2.

The corresponding sequence of $\mathcal{NM}$-matrices of G is given in Fig. 3. Note that when $k = 3$, the matrix $\mathcal{NM}^{\{3\}}(G)$ has no zero entry and hence the iteration number $k(G) := 3$.

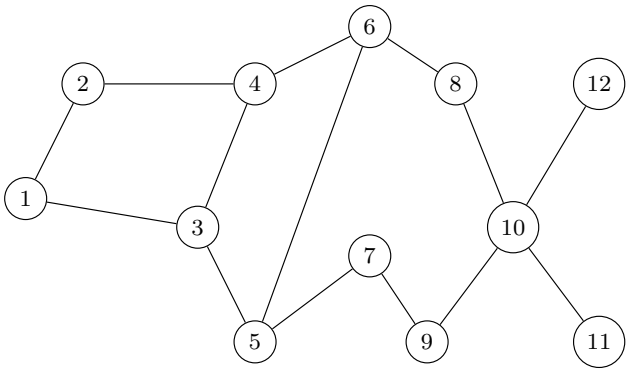

Fig. 2 A graph G on 12 vertices and 14 edges

$$\begin{pmatrix}
-2 & 2 & 3 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
2 & -2 & -2 & 3 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0\\
2 & -2 & -3 & 3 & 3 & -2 & -1 & 0 & 0 & 0 & 0 & 0\\
-2 & 2 & 3 & -3 & -2 & 3 & 0 & -1 & 0 & 0 & 0 & 0\\
-1 & 0 & 3 & -2 & -3 & 3 & 2 & -1 & -1 & 0 & 0 & 0\\
0 & -1 & -2 & 3 & 3 & -3 & -1 & 2 & 0 & -1 & 0 & 0\\
0 & 0 & -1 & 0 & 3 & -1 & -2 & 0 & 2 & -1 & 0 & 0\\
0 & 0 & 0 & -1 & -1 & 3 & 0 & -2 & -1 & 4 & -1 & -1\\
0 & 0 & 0 & 0 & -1 & 0 & 2 & -1 & -2 & 4 & -1 & -1\\
0 & 0 & 0 & 0 & 0 & -1 & -1 & 2 & 2 & -4 & 1 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 4 & -1 & -1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 4 & -1 & -1
\end{pmatrix}$$

(a) $\mathcal{NM}^{\{1\}}(G)$

$$\begin{pmatrix}
-4 & 2 & 3 & 3 & 5 & -4 & -2 & -2 & -1 & 0 & 0 & 0\\
2 & -4 & 3 & 3 & -4 & 5 & -2 & -2 & 0 & -1 & 0 & 0\\
1 & 1 & -6 & 2 & 3 & 3 & 3 & -3 & -2 & -2 & 0 & 0\\
1 & 1 & 2 & -6 & 3 & 3 & -3 & 5 & -2 & -2 & -1 & -1\\
2 & -4 & 2 & 2 & -7 & 3 & 2 & 4 & 4 & -4 & -2 & -2\\
-4 & 2 & 2 & 2 & 3 & -7 & 2 & 4 & -4 & 4 & -2 & -2\\
-2 & -2 & 4 & -3 & 4 & 4 & -5 & -4 & 4 & 4 & -2 & -2\\
-2 & -2 & -3 & 4 & 4 & 4 & -4 & -7 & 2 & 2 & 1 & 1\\
-1 & 0 & -2 & -2 & 5 & -4 & 3 & 3 & -6 & 2 & 1 & 1\\
0 & -1 & -2 & -2 & -4 & 5 & 3 & 3 & 2 & -6 & 1 & 1\\
0 & 0 & 0 & -1 & -2 & -2 & -2 & 4 & 3 & 3 & -4 & 1\\
0 & 0 & 0 & -1 & -2 & -2 & -2 & 4 & 3 & 3 & 1 & -4
\end{pmatrix}$$

(b) $\mathcal{NM}^{\{2\}}(G)$

$$\begin{pmatrix}
-8 & 2 & 2 & 4 & 4 & 4 & 4 & 4 & 4 & -8 & -6 & -6\\
2 & -8 & 2 & 4 & 4 & 4 & 4 & 4 & -8 & 4 & -6 & -6\\
1 & 1 & -9 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & -7 & -7\\
1 & 1 & 1 & -11 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & -11 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & -11 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & -11 & 1 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1 & 1 & 1 & -11 & 1 & 1 & 1 & 1\\
2 & -8 & 2 & 2 & 2 & 2 & 2 & 2 & -10 & 2 & 1 & 1\\
-8 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & -10 & 1 & 1\\
-6 & -6 & -7 & 4 & 4 & 4 & 4 & 4 & 3 & 3 & -8 & 1\\
-6 & -6 & -7 & 4 & 4 & 4 & 4 & 4 & 3 & 3 & 1 & -8
\end{pmatrix}$$

(c) $\mathcal{NM}^{\{3\}}(G)$

Fig. 3 Sequence of $\mathcal{NM}$ matrices corresponding to G

2 Methodology

In this section, we state some important results connecting the concept of distance between any two vertices with the entries of the matrix $\mathcal{NM}^{\{l\}}$, $1 \le l \le k$. The proofs of the mathematical results from this section are in "Appendix". These results will be useful for the algorithms stated in the next section.

2.1 Mathematical foundation

Theorem 3 *Let G be an undirected unweighted simple graph on n vertices and let $k(G)$ be the iteration number of G. If $\eta_{ij}^{\{l\}} < 0$ for some $1 \le l \le k(G)$, then*

1. $\eta_{ij}^{\{p\}} = 0, 1 \le p \le l-1$ *and* $\eta_{ij}^{\{q\}} > 0, l+1 \le q \le k(G)$. *Converse is also true.*
2. *There exists a vertex x such that $\eta_{ix}^{\{l\}} > 0$ and $\eta_{xj}^{\{l\}} > 0$. Converse need not be true.*

Example For the graph G in Fig. 2 and its matrices in Fig. 3, let us consider the vertices $i = 2$ and $j = 5$. Note that $\eta_{2,5}^{\{2\}} = -4 < 0$. Hence $\eta_{2,5}^{\{1\}} = 0$ and $\eta_{2,5}^{\{3\}} = 4 > 0$.

Theorem 4 *Let G be an undirected unweighted simple graph on n vertices and let $k(G)$ be the iteration number of G. For $1 \le l \le k(G)$, the off-diagonal elements of $\mathcal{NM}^{\{l\}}$ can be characterized as follows, for $1 \le i \ne j \le n$*

1. $\eta_{ij}^{\{l\}} = 0$ *if and only if* $d_G(i,j) > 2^l$.
2. $\eta_{ij}^{\{l\}} > 0$ *if and only if* $0 < d_G(i,j) \le 2^{l-1}$.
3. $\eta_{ij}^{\{l\}} < 0$ *if and only if* $2^{l-1} < d_G(i,j) \le 2^l$.

Example For the graph G in Fig. 2 and its associated matrices in Fig. 3, let the vertices be $i = 2$ and $j = 5$. Note that the distance between 2 and 5 is $3 < 2^2$ and that $\eta_{2,5}^{\{2\}} = -4 < 0$, where $l = 2$. Similarly, $\eta_{1,10}^{\{3\}} \ne 0$ and $2^2 < d(1,10) = 5 \le 2^3$.

Theorem 5 *Let G be an undirected unweighted simple graph on n vertices and let $k(G)$ be the iteration number of G. If $\eta_{ij}^{\{l\}} = -1$, for $1 \le l \le k(G)$, then $d_G(i,j) = 2^l$.*

Remark 5 Note the converse of the above theorem need not hold always. For example, consider the cycle C_4 and its neighbourhood matrix $\mathcal{NM}(C_4)$ for $l = 1$. For non-adjacent vertices i and j, $\eta_{ij}^{\{1\}} = -2$ while $d_{C_4}(i,j) = 2$.

From the above theorem, we define the parameter called range of distance between any pair of vertices in a graph as follows:

Definition 4 Given two vertices i and j, $rd_G(i,j)$, is defined as follows:

$$rd_G(i,j) = \begin{cases} 1, & \text{if } (i,j) \in E(G) \\ p, & \text{if } i,j \text{ are connected and} \eta_{ij}^{\{p\}} < 0 \\ \infty, & \text{if } i,j \text{ are not connected} \end{cases}$$

Remark 6 For any two connected non-adjacent vertices $i,j \in V(G)$, we have $2^{rd_G(i,j)-1} < d_G(i,j) \leq 2^{rd_G(i,j)}$

Theorem 6 *A graph G is connected if and only if the corresponding matrix $\mathcal{NM}^{\{k\}}$ has no zero entries. In addition, the iteration number k(G) of the graph $G \neq K_n$ is given by $k(G) = \lceil \log_2(\text{diameter}(G)) \rceil$. (For $G = K_n, k(G) = 1$ and $\mathcal{NM}^{\{1\}} = \mathcal{NM}(G) = -C(G)$)*

Corollary 1 *A graph G is disconnected if and only if $z < n^2$. Further, the iteration number k(G) of G is given by $k(G) = \lceil \log_2 S \rceil$, where S is the maximum over the diameter of components of G.*

Remark 7 For any connected graph G, it is immediate from the above discussion that $1 \leq k(G) \leq \lceil \log_2(n-1) \rceil$, that is, the iteration number is maximum when diameter(G) is maximum. It is well-known that the path graph on n vertices has the largest diameter among all graphs on n vertices (with diameter $n-1$), while any graph with diameter at most 2 has $k(G) = 1$.

2.2 An efficient shortest path algorithm

In this section, we present a brief description of the proposed algorithm to find the shortest path between any pair of vertices in a given graph G along with the pseudocodes in *MATLAB* of the algorithm.

2.2.1 Description of the algorithm

The objective of this work is to reduce the computational time involved in finding the shortest path between any given pair of vertices. The algorithm is based on the construction of the sequence of graphs and its corresponding matrices $\mathcal{NM}^{\{l\}}, 1 \leq l \leq k(G)$. Using this sequence of matrices and the results from the previous section, we find the shortest path and also show that the computational time taken is very less for graphs of large size. The algorithm is presented in three modules:

In the first module (Algorithm 1), we compute the $\mathcal{NM}(G)$ matrix and the sequence of $k(G)$ matrices, namely powers of $\mathcal{NM}(G)$ [denoted by $SPG(:,:,:)$] associated with G. The algorithm is named as SP-$\mathcal{NM}(A)$, where A is the adjacency matrix of G.

Algorithm 1 SEQUENCE OF POWERS OF $\mathcal{NM}$ MATRIX CORRESPONDING TO THE GIVEN GRAPH.

Objective: To find the iteration number k and construct the sequence powers of $\mathcal{NM}$ matrix of a given graph G, that is $\mathcal{NM}^{\{l\}}$, for every $l, 1 \leq l \leq k$.
Input: Adjacency matrix A of a undirected unweighted simple graph G.
Output: The number of vertices n, iteration number k and $SPG-$ A three dimensional matrix of size $(n \times n \times k)$, that is for each l, $1 \leq l \leq k$, the $n \times n - \mathcal{NM}^{\{l\}}$ matrix is given by $SPG(:,:,l)$.

```
1: procedure [n, k, SPG] = SP−NM(A)
2:     l ← 1;
3:     n ← Number of rows or columns of matrix A
4:     b ← 0                                              ▷ Initialize
5:     while (True) do
6:         L ← Laplacian matrix of A
7:         NM ← A × L                          ▷ Construct the NM matrix from A.
8:         SPG(:,:,l) ← NM                     ▷ SPG-sequence of powers of NM matrix.
9:         a ← nnz(NM)                         ▷ nnz-Number of non zero entries
10:        if a == n^2 then k ← l break
11:        else if isequal(a, b) == 1 then k ← l − 1 break
12:        else b ← a
13:            A ← NM; A(A ≠ 0) ← 1;
14:            for p ← 1 to n do A(p, p) ← 0
15:            end
16:            l ← l + 1
17:        end
18:    end
```

In the main module (Algorithm 2), given a graph G, SPG array and a pair of vertices i and j, the shortest path between i and j is obtained with recursive calls to the module CN and itself. The name of the algorithm is $\mathcal{NM}-$ ShortestPath(i, j, k, SPG).

Here, first we check if the vertices i and j are connected and estimate the range of distance between i and j, that is, we find $l := rd_G(i,j)$ such that, $2^{l-1} < d_G(i,j) \leq 2^l$ by applying the idea from Theorems 3 and 4. In particular, we find least l, for which $\eta_{ij}^{\{l\}} \neq 0$. This is computed through steps 1–6 of Algorithm 2.

Algorithm 2 Shortest path between given pair of vertices

Objective: To find the shortest path between i and j using the sequence of powers of $\mathcal{NM}$ matrix.
Input: $i, j \in V$, iteration number k and sequence of $\mathcal{NM}$ matrix SPG from **Algorithm 1**.
Output: S_P- Shortest path

```
1: procedure S_P = NM-ShortestPath(i, j, k, SPG)
2:     if isequal(i, j) == 1 then S_P ← 0
3:     else if (SPG(i, j, k) == 0) then S_P ← ∞
4:     else l ← 1
5:         while (SPG(i, j, l) == 0) do l ← l + 1
6:         end
7:         if l == 1 then
8:             if SPG(i, j, 1) > 0 then S_P ← [i, j]
9:             else v ← positions(SPG(i, :, 1) > 0)
10:                y ← Any one position of (v(positions(SPG(v, j, 1) > 0))   ▷ y ∈ N_G(i) ∩ N_G(j)
11:                S_P ← [i, y, j]
12:            end
13:        else y ← CN(i, j, l, SPG)
14:            si ← 1
15:            S_P ← [i, y, j]
16:            nt ← 1
17:            while (nt < l) do
18:                Y_P ← ∅
19:                va ← 0
20:                for p2 ← 1 to (2 × si) do
21:                    i ← S_P(p2)
22:                    j ← S_P(p2 + 1)
23:                    if SPG(i, j, 1) > 0 then Y_P ← [Y_P, i]
24:                    else if SPG(i, j, 1) < 0 then v ← positions(SPG(i, :, 1) > 0)
25:                        y ← Any one vertex of (v(positions(SPG(v, j, 1) > 0))   ▷ y ∈ N_G(i) ∩ N_G(j)
26:                        Y_P ← [Y_P, i , y ]
27:                    else rr ← 2
28:                        while (SPG(i, j, rr) == 0) do
29:                            rr ← rr + 1
30:                        end
31:                        y ← CN(i, j, rr, SPG)
32:                        Y_P ← [Y_P, i , y]
33:                        va ← va + 1
34:                    end
35:                end
36:                S_P ← [Y_P, S_P((2 × si) + 1) : end)]
37:                si ← va
38:                nt ← nt + 1
39:            end
40:        end
41:    end
```

When $l = 1$: If $\eta_{ij}^{\{1\}} > 0 \Rightarrow d_G(i,j) = 1$, then $< i, j >$ is the shortest path and if $\eta_{ij}^{\{1\}} < 0 \Rightarrow d_G(i,j) = 2$, then we have $< i, x, j >$, where x is a common neighbour of i and j, to be the shortest path. This is computed through steps 7–12 of Algorithm 2.

Algorithm 3 Common neighbour of given pair of vertices in $G^{\{l\}}$

Objective: To find the vertex $x \in V$ which satisfies $d_G(i,y) = 2^{l-1}$ and $d_G(y,j)$ is minimum.
Input: $i, j \in V$, l from **Definition 4** and sequence of $\mathcal{NM}$ matrix SPG from **Algorithm 1**.
Output: $y \in V$

```
1: procedure y =CN(i, j, l, SPG)
2:     X ← positions(SPG(i, :, l − 1) < 0)
3:     X ← X − {i}
4:     ct ← 1; r1 ← SPG(X, j, ct)
5:     while (nnz(r1) == 0) do                ▷ nnz- Number of non zero entries
6:         ct ← ct + 1
7:         r1 ← SPG(X, j, ct)
8:     end
9:     if ct == 1 then
10:        if |positions(r1 > 0)| > 0 then y ← X( any position of r1 > 0)
11:        else y ← X(any position of minimum of r1 )
12:        end
13:    else Z ← X(positions(r1 < −1))
14:        if |Z| == 1 then y ← Z
15:        else if |Z| == 0 then y ← X( any position of r1 == −1)
16:        else
17:            for c ← 1 : |Z| do
18:                i ← Z(c); j ← j
19:                D ← 0
20:                if isequal(i, j) == 1 then D1 ← 0
21:                else
22:                    while (True) do rr ← 1
23:                        while (SPG(i, j, rr) == 0) do
24:                            rr ← rr + 1
25:                        end
26:                        if rr == 1 then
27:                            if SPG(i, j, rr) > 0 then D1 ← 1 break
28:                            else D1 ← 2 break
29:                            end
30:                        else if (SP(i, j, rr) == −1) then D1 ← 2^rr break
31:                        else D ← 2^(rr−1) + D
32:                            while (True) do y ← CN ( i, j, rr, SPG )
33:                                i ← y break
34:                            end
35:                        end
36:                    end
37:                end
38:                SPD ← D + D1
39:                SO(c) ← SPD
40:            end
41:            [x1 x2] ← min(SO)
42:            y ← Z(x2)
43:        end
44:    end
```

For $l \geq 2$, we proceed to find a vertex y such that $d_G(i,y) = 2^{l-1}$ and $d_G(y,j)$ is minimum by invoking the subroutine stated as Algorithm 3. In the module (Algorithm 3), we find the vertex y such that $d_G(i,y) = 2^{l-1}$ and $d_G(y,j)$ is minimum. The algorithm is named as CN(i, j, l, SPG). Now, for $l \geq 2$, we have $\eta_{ij}^{\{l\}} < 0$ and hence we find the set $X = \{x : \eta_{ix}^{\{l-1\}} < 0\}$, where x is the vertex such that $2^{l-2} < d_G(i,x) \leq 2^{l-1}$ and $d_G(x,j) \leq 2^{l-1}$ (Theorem 4). From this set, we want to choose one y such that $d_G(i,y) = 2^{l-1}$ and $d_G(y,j) \leq 2^{l-1}$. Compute $ct := \min\{rd_G(x,j) : x \in X\}$. Note that $1 \leq ct \leq l-1$. Let $Z = \{x \in X : rd_G(x,j) = ct\}$. Recursively, we calculate the distance between vertices in Z and j. Let y be the vertex such that, $d_G(y,j) \leq d_G(z,j)$, for all $z \in Z$, that is the distance $d_G(y,j)$ is minimum and that y satisfies $d_G(i,y) = 2^{l-1}$ and $d_G(y,j)$ is minimum.

Upon repeated application of this iterative process to the pairs $\{i, y\}$ and $\{y, j\}$, we get all the vertices lying on the shortest path between i and j. (rest of Algorithm 2)

In general, the algorithm requires computation of $k(G)$ matrices and storing it for further purpose, hence there is a huge requirement of storage space and the space complexity spans $O(k(G)n^2)$.

Example Consider the graph in Fig. 2 and having computed the matrices in Fig. 3 using Algorithm 1, let us illustrate Algorithms 2 and 3 for $i = 2$ and $j = 11$.

First we find that $\eta_{2,11}^{\{3\}} = SPG(2, 11, 3) = -6 \neq 0$. Therefore, $l = rd(2, 11) = 3$. Since $l \geq 2$, we find a vertex y by invoking Algorithm 3, such that $d_G(2, y) = 4$ and $d_G(y, 11)$ is minimum. First, we get the set $X = \{5, 7, 8, 10\}$ by scanning the second row of $\mathcal{NM}^{\{2\}}(G) := SPG(2, :, 2)$.

Next, we compute $ct = \min\{rd(x, 11) : x \in X\} = 1$ using steps 5–7 and $r1$ stores those vertices from X which had $ct = 1$. Since $ct = 1$, by step 10, choosing positions of $SPG(r1, 11, 1) > 0$ directly, we get $y = 10$ and the distance between 10 and 11 is 1. Hence, we get the vector $S_P = [2\ 10\ 11]$ as the output at this step.

Now recursively, repeating the procedure for the vertex pair (2, 10): $\eta_{2,10}^{\{2\}} = SPG(2, 10, 2) = -1$, we have $l = rd(2, 10) = 2$. Since $l = 2$, we find a vertex y by invoking Algorithm 3. Here, $X = \{3, 6\}$ is obtained by scanning the second row of $\mathcal{NM}^{\{1\}}(G) := SPG(2, :, 1)$. From steps 5–7, compute $ct = 1$and by steps 9–11, we get $y = 6$, with distance between 6 and 10 equals 2. So we get the vector [2 6 10 11].

Now repeating the process with the vertex pairs (2, 6) and (6, 10), we get [2 4 6] and [6 8 10]. Finally, combining the solutions, we get [2 4 6 8 10 11] as the shortest path.

2.2.2 Time complexity

As stated before, the proposed algorithm has three modules and first, we discuss their time complexity individually.

1. In the first module (Algorithm 1), we compute the sequence of $k(G)$ matrices, $\mathcal{NM}^{\{k\}}(G)$. The running time of this module is $\mathcal{O}([k+1]n^{2.3737})$ as we apply the product of two matrices namely adjacency A(G) and the Laplacian $C(G)$ to obtain the $\mathcal{NM}(G)$ and we do this for $k(G)$ times, for where $k(G) = \lceil \log_2(diameter(G)) \rceil$, while G is connected and $k(G) + 1$times while G is disconnected. We use Coppersmith–Winograd algorithm [6] for matrix multiplication which has a $\mathcal{O}(n^{2.3737})$ running time.
2. For the module in Algorithm 3, the worst case running time is $\mathcal{O}(k^2 n)$, where $k(G) \leq \lceil \log_2(diameter(G)) \rceil \leq \lceil \log_2 n \rceil$ and the best case running time is $\mathcal{O}(n)$.
3. For the main module (Algorithm 2) $\mathcal{NM}$–ShortestPath(i, j, k, SPG), the best case running time is $\mathcal{O}(n)$ when $k = 1$ while the worst case running time of this algorithm is $\mathcal{O}(2^k k^2 n)$, $k(G) = \lceil \log_2(diameter(G)) \rceil \leq \lceil \log_2 n \rceil$, which turns out to be $\mathcal{O}[(n \log_2 n)^2]$. Note that for most of the choices of i and j, whose distance lies between $2^{l-1} < d(i, j) \leq 2^l$ for $l \leq k$, this algorithm runs only in $\mathcal{O}(2^l l^2 n), 1 \leq l \leq k$.

Given any graph G and a pair of vertices i and j in G, the computational time required by our algorithms in the worst case is $\mathcal{O}([k+1]n^{2.3737}) + \mathcal{O}(2^k \cdot k^2 \cdot n)$, where the computation of $\mathcal{NM}^{\{l\}}$ is a one-time procedure for each graph. In addition, to find all possible shortest paths from a single source node i, we require $\mathcal{O}([k+1]n^{2.3737}) + \mathcal{O}(2^k \cdot k^2 \cdot n(n-1))$, k is the iteration number of G. Moreover, for different pairs of vertices within the same graph, the bound given here is too large.

As it is well-known, the notion of the running time of the algorithm always refer to worst-case running time, which do not always resemble the algorithms' behaviour on real-life instances. However, algorithms with large worst-case running times do not inevitably perform poorly in practice.

Hence, in the next section, we empirically analyse the algorithms on various datasets to justify the bound.

3 Results and discussion

In this section, we consider various datasets of graphs and illustrate the results from previous section.

For this purpose, we use special graph classes and compare the proposed shortest path algorithm with the existing algorithms such as Dijkstra, BFS and Bellman–Ford to find the shortest path between any given pair of vertices. We have tested our algorithm on various randomly generated collection of graphs using MATLAB.

We would like to recall from Theorem 6 that the iteration number for any connected graph is given by $1 \leq k(G) \leq \lceil \log_2(diameter(G)) \rceil \leq \lceil \log_2(n-1) \rceil$. Since path graph has diam $n-1$ and requires computation of $\lceil \log_2(n-1) \rceil - \mathcal{NM}^{\{l\}}$ matrices, maximum number of iterations in the worst case. Hence, to ensure connectivity, sparseness and the worst cases possible, we sample our datasets by adding randomly edges to a path graph P_n.

On the other hand, it is well-established fact in the complex network analysis that any real-world network exhibits sparseness and a behaviour of small world network whose average diameter is approximately 6. In this regard, we have also found that all the considered real-world networks required computation of 3 or 4 of the $\mathcal{NM}^{\{l\}}$ matrices giving us the iteration number as 3 or 4.

We have experimentally tested our theory on various datasets. However, here for brevity, we present four such datasets.

DS_1 : This dataset contains 1100 graphs where each graph is connected and contains 100 vertices. The first

graph of this collection is $P_{100} \cup \{e\}$, where e is any edge. Consecutively, a graph $G_m \in DS_1$, for $1 \leq m \leq 1100$ contains $P_{100} \cup \{e_1, e_2, \ldots, e_m\}$ for a random generated set of m–distinct edges. In particular, G_m contains 100 vertices and $99 + m$ edges.

DS_2 : Similar to the dataset DS_1, this dataset contains 1100 graphs where each graph is connected and has 200 vertices. The first graph of this collection is $P_{200} \cup \{e\}$, where e is any edge. A graph $G_m \in DS_2$, for $1 \leq m \leq 1100$ contains $P_{200} \cup \{e_1, e_2, \ldots, e_m\}$ for a randomly generated set of m–distinct edges. In particular, G_m contains 200 vertices and $199 + m$ edges.

DS_3 : This dataset also contains 1100 graphs, where each graph is connected and has 300 vertices. The first graph of this collection is $P_{300} \cup \{e\}$, where e is any arbitrary edge. A graph $G_m \in DS_3$, for $1 \leq m \leq 1100$ contains $P_{300} \cup \{e_1, e_2, \ldots, e_m\}$ for a randomly generated set of m–distinct edges. In particular, G_m contains 300 vertices and $299 + m$ edges.

Further, to implement our algorithm and test our claims on a large-scale real-world network, we have considered the Facebook graph obtained from SNAP [17].

DS_4 : This dataset contains one graph with 4039 vertices, 88,234 edges and has a diameter of 8. (Hence, we see that the iteration number is 3).

Figures 4, 5 and 6 represent the total time required to compute the shortest path between distinct pairs of vertices for each of the graphs given in the considered datasets DS_1, DS_2, DS_3, respectively, by all the four shortest path algorithms, namely $\mathcal{NM}$ shortest path algorithm, BFS, Bellman–Ford and Dijkstra.

Figure 4 represents the computational time for G_{20} to G_{1100}, while the inner box of Fig. 4 contains the computational time required for G_1 to G_{50} to compute the shortest path between all pairs of vertices. As the scale has a huge variation at the beginning, to visualize the plots and to give a clear distinction of the computational time required by all the four algorithms, the plots are presented this way. It is immediately seen that the computational time of $\mathcal{NM}$-Shortestpath algorithm for (nearly) path graphs is too large and the plots become difficult to visualize at the beginning. From this figure, we observe that the running time of the $\mathcal{NM}$ shortest path algorithm decreases exponentially and rapidly, with increase in number of edges in the graph. There is a steep and sharp peak at the beginning, which is due to the fact that iteration number is large for graphs which are almost path graphs.

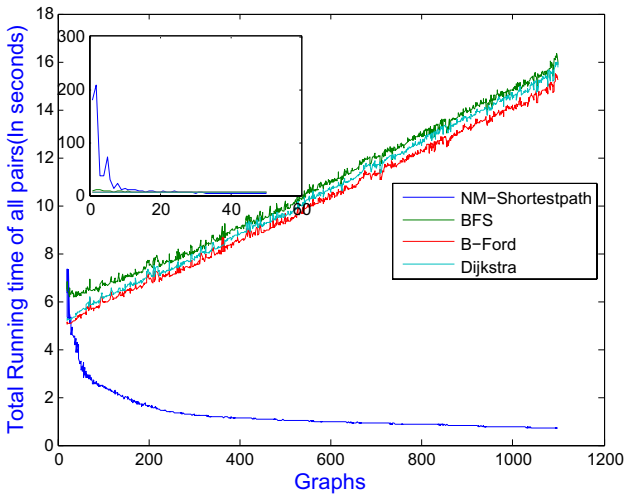

Fig. 5 Running times for dataset DS_2

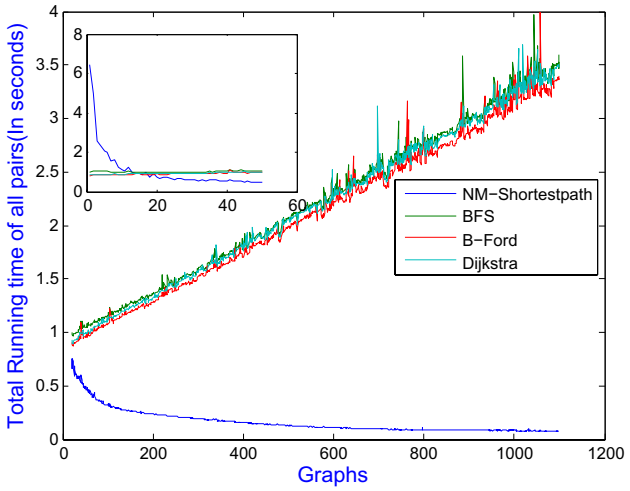

Fig. 4 Running times for dataset DS_1

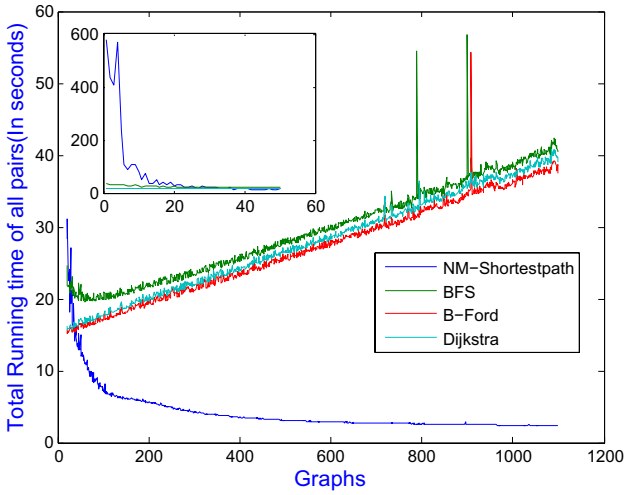

Fig. 6 Running times for dataset DS_3

With increasing number of edges, since the graphs are connected, the graph's diameter decreases.

In particular, we have that from G_{20} to G_{1100} in DS_1, as the diameter of the graph decreases, the iteration number $k(G_m)$ also decreases as m runs from 20 to 1100. Hence, the computation of sequence of $\mathcal{NM}$ matrices and the shortest path between any pair of vertices decreases, while the other three algorithms steadily increases with the increase in number of edges in the graph. However, we conclude that less computational time is required by $\mathcal{NM}$–Shortest-Path when compared to the existing algorithms.

A similar behaviour is observed with the other two datasets DS_2 and DS_3 when the computational time of the four algorithms is compared. In fact, a similar pattern of plots are obtained which is evident in Figs. 5 and 6. Hence, we conclude that our algorithm performs extremely well when the graph is not too sparse specifically, not a nearly path graph. In fact, on an average, we find that for the graphs with edge density greater than 25%, our proposed algorithm runs correctly with very high efficiency.

Further, Fig. 7 represents the comparative analysis of the computational time of the proposed algorithm for the considered datasets DS_1, DS_2, DS_3. Here, we observe that as n increases, the average computational time increases but decreases exponentially as the density of the graphs increases. Moreover, they form a similar pattern showing that irrespective of the order of the graph as the density of the graph increases and diameter of the graph decreases, the running time is comparatively minimal. This leaves a positive note for studying dense graphs in minimal amount of time.

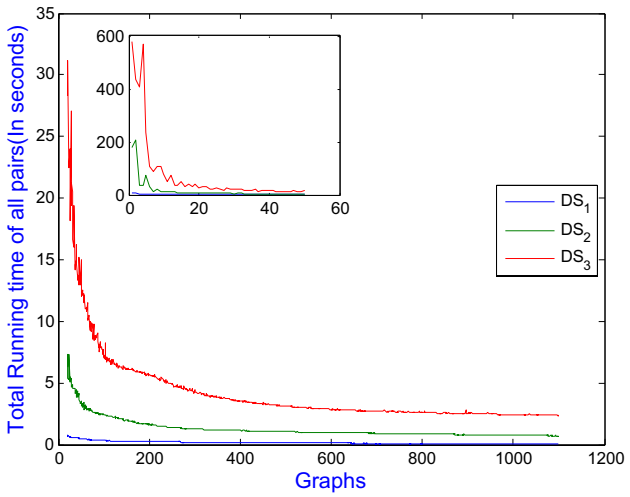

Fig. 7 Time taken by $\mathcal{NM}$ Shortestpath on different datasets DS_1, DS_2 and DS_3

For the Facebook graph considered in DS_4, we have $k(G) = 3$, that is, the graph is connected and has diameter 8. As there are 8,158,780 possible pairs of vertices, we first randomly choose a set of pairs to test the efficiency of the existing algorithms, namely the Dijkstra, Bellman–Ford and BFS with the proposed $\mathcal{NM}$-ShortestPath algorithm. Tables 1 and 2 represent the total running time of the algorithms to compute the shortest path between randomly chosen pairs of vertices, namely 50,000 pairs, 100,000 pairs, all the pairs of vertices which are at maximum distance from each other, that is, the diametrically opposite pairs of vertices (there were 15,620 such pairs) and finally, we considered all the possible pairs of vertices (namely the 81,587,80 pairs).

In Table 1, the data represents the total computational time required and the mean computational time per pair for the three existing algorithms. Table 2 represents the runtime of $\mathcal{NM}$ shortest path algorithm, where the run time for computing the sequence of matrices and the shortest paths are given in columns 2 and 3 along with total computational time, given in column 4. Here, we observe that there is a overhead computational time (of approx 115 s) involved to compute all the k-matrices of $\mathcal{NM}$ sequence. Having computed that finding the shortest path takes less than 0.0003 s per pair.

From the tables, it is evident that for 50,000 pairs of vertices, the $\mathcal{NM}$ shortest path algorithm takes only 155.5888 s including 115.9333 s to compute the sequence

Table 2 Computation time (in s) of the $\mathcal{NM}$ shortest path algorithm on DS_4

No. of pairs	SP-$\mathcal{NM}$ $\mathcal{NM}^{(l)}, l \leq 3$	$\mathcal{NM}$-Shortest Path-module	Total
50,000	115.9333	39.6555	155.5888
100,000	118.0746	76.8048	194.8794
15,620	118.6954	9.5945	128.2899
8,158,780	122.1065	3004.0527	3126.1592

Table 1 Computation time (in s) of the three known shortest path algorithms on DS_4

No. of pairs	Dijkstra		Bellman–Ford		BFS	
	Total time	Mean	Total time	Mean	Total time	Mean
50,000	3992.4677	0.0798	4705.6922	0.0941	3840.4749	0.0768
100,000	7982.6267	0.0798	9295.2256	0.093	7665.1566	0.0767
15,620	1250.3008	0.08	1527.5449	0.0978	1203.2897	0.077

of $\mathcal{NM}$ matrices as preprocessing and 39.6555 s to compute the actual shortest path between the given pairs. While for the same pairs, the BFS algorithm has taken 3840.4749 s, Dijkstra's takes 3992.4677 s and Bellman–Ford requires a maximum time of 4705.6922 s.

Similarly, for the 100,000 randomly chosen pairs of vertices, we get that $\mathcal{NM}$ shortest path algorithm takes 194.8794 s only, while the other algorithms takes around 7600 to 9300 s.

Now considering the pairs of diametrically opposite vertices, we anticipate that the running time for the $\mathcal{NM}$ shortest path algorithm will be large for such pairs when compared to other possible pairs. Row 3 in Tables 1 and 2 represents the running time for these 15,620 pairs of vertices which have a shortest path distance exactly 8. Our expectations were justified and proven to be true as we can see that the mean computational time for a pair of vertices with diameter 8 takes maximum time of (0.0082), while the average is 0.0004 over all the possible pairs.

Row 4 represents the total running time to compute all-pairs shortest path of the graph. As we can see the $\mathcal{NM}$ shortest path algorithm takes only 3126.1592 s (which is less than an hour) to find all pair shortest path or an average 0.0004 s to compute the running time of any given pair in the graph. It is immediate to see that the running time of existing shortest path algorithms for all pairs takes more than 158 h (estimated based on the mean time obtained for random pairs collection).

The above plots and tables clearly indicate that the proposed $\mathcal{NM}$ shortest path algorithm is efficient in terms of accuracy and time involved than the existing shortest path algorithms. Moreover, for increasing size of graphs, the difference in the running time of an existing algorithm and our algorithm is also very high, with our algorithm performing well almost every time. The only drawback is that when the graph's diameter is close to n, the computation of the sequence of matrices takes more time and also finding shortest path takes considerably larger amount of time.

4 Conclusion

In this paper, our focus is to design an efficient algorithm to find the shortest path between any given pair of vertices in a graph. To achieve this, we proposed the concept of sequence of powers of $\mathcal{NM}(G)$ matrix that can be associated with a graph to reveal more information. With the help of these matrices, we presented an efficient algorithm ($\mathcal{NM}$-ShortestPath) to compute the shortest path on any graph G, with the computational time of $\mathcal{O}([k+1]n^{2.3737}) + \mathcal{O}(2^k \cdot k^2 \cdot n)$, in the worst case, where the computation of $\mathcal{NM}^{\{l\}}$ is a one-time procedure for each graph. Further, for different pairs of vertices within the same graph, the bound given here is found to be too large.

An empirical study of the algorithm was undertaken and found to be satisfying about the efficiency and accuracy of the same. Thus, demonstrated that $\mathcal{NM}$-ShortestPath algorithm is expeditious and highly effective than the existing algorithms when the graph has small diameters. The only drawback is that when the graph's diameter is close to n, the computation of the sequence of matrices takes more time and also finding shortest path takes a considerably larger amount of time. Further, in real-life instances, the networks pertain to simulate to be "Small-World" with very small diameter when compared to the size of the network and hence the proposed algorithm performs very well in such cases.

4.1 Future directions

In this paper, only undirected unweighted graphs were considered. As a future direction, one could extend this technique to weighted graphs and directed graphs. Also from a theoretical perspective, there is a lot of scope to explore and study the matrix properties of $\mathcal{NM}$ such as the spectral analysis from both theoretical and computational aspects. Further, with the association of such a sequence of matrices, we conjecture that many graph related problems could be solved with less time complexity.

Acknowledgements The authors would like to acknowledge and thank DST-SERB Young Scientist Scheme, India (Grants No. SB/FTP/MS-050/2013), for their support to carry out this research at SRM Research Institute, SRM Institute of Science and Technology. Mr. Sivakumar Karunakaran would like to thank SRM Research Institute and the Department of Mathematical Science, IIT (BHU), for their support during the preparation of this manuscript. He would also like to thank Mr. Ashish Kumar Maurya and Mr. Amit Biswas, research scholars of IIT (BHU) for fruitful discussions.

Compliance with ethical standards

Competing interest The authors declare that they have no competing interest.

Appendix

In this section, we present all the necessary proofs for the theorems discussed in Sect. 2.

Proof of Theorem 3

Theorem 3 Let G be an undirected unweighted simple graph on n vertices and let $k(G)$ be the iteration number of G. *If* $\eta_{ij}^{\{l\}} < 0$ for some $1 \leq l \leq k(G)$, then

1. $\eta_{ij}^{\{p\}} = 0, 1 \le p \le l-1$ and $\eta_{ij}^{\{q\}} > 0, l+1 \le q \le k(G)$. Converse is also true.
2. There exists a vertex x such that $\eta_{ix}^{\{l\}} > 0$ and $\eta_{xj}^{\{l\}} > 0$. Converse need not be true.

Proof

1. For some, l $1 \le l \le k(G)$, given $\eta_{ij}^{\{l\}} < 0 \implies$ by the Definition 1, i and j are not adjacent in $G^{\{l\}}$. In fact, i and j are not adjacent in $G^{\{p\}}$, for any $p, 1 \le p \le l$, that is, $a_{ij}^{\{p\}} = 0$. By the Definition 1, $a_{ij}^{\{p\}} = 0$ implies $\eta_{ij}^{\{p-1\}}$ must have been zero. Hence, we have $\eta_{ij}^{\{p\}} = 0$, for $1 \le p \le l-1$. Further, $\eta_{ij}^{\{l\}} < 0 \implies a_{ij}^{\{l+1\}} = 1$, that is, i and j become adjacent in $G^{\{l+1\}}$. Hence, $\eta_{ij}^{\{l+1\}} > 0$ (By Definition 1). In addition, we have $G^{\{l+1\}}$ is a subgraph of $G^{\{q\}}$ for every $q \ge l+1$. Hence by definition of $\mathcal{NM}(G)$, $\eta_{ij}^{\{q\}} > 0$, for every $q, q \ge l+1$.

   Conversely, suppose $\eta_{ij}^{\{p\}} = 0, 1 \le p < l$ and $\eta_{ij}^{\{q\}} > 0$, $q \ge l+1$ holds, then in particular, for $p = l-1$ and for $q = l+1$, we have,

$$\eta_{ij}^{\{l-1\}} = 0 \implies a_{ij}^{\{l\}} = 0 \implies \eta_{ij}^{\{l\}} \le 0 \tag{1}$$

$$\eta_{ij}^{\{l+1\}} > 0 \implies a_{ij}^{\{l+1\}} = 1 \implies \eta_{ij}^{\{l\}} \ne 0 \tag{2}$$

   Equations (1) and (2) together implies $\eta_{ij}^{\{l\}} < 0$.
2. If $\eta_{ij}^{\{l\}} < 0$, then by the Definition 1, i and j are not adjacent in $G^{\{l\}}$ and there exist $|N_{G^{\{l\}}}(i) \cap N_{G^{\{l\}}}(j)|$ common neighbours between i and j in $G^{\{l\}}$. Let x be any such common neighbour of i and j in $G^{\{l\}}$, then $\eta_{ix}^{\{l\}} > 0$ and $\eta_{xj}^{\{l\}} > 0$.

   To disprove the converse, consider the case when three vertices $\{i, x, j\}$ are mutually adjacent in $G^{\{l\}}$, then by Definition 1 we have, $\eta_{ix}^{\{l\}} > 0, \eta_{xj}^{\{l\}} > 0$ and $\eta_{ij}^{\{l\}} > 0$ which contradicts the conclusion $\eta_{ij}^{\{l\}} < 0$. □

Proof of Theorem 4

Theorem 4 Let G be an undirected unweighted simple graph on n vertices and let $k(G)$ be the iteration number of G. For $1 \le l \le k(G)$, the off-diagonal elements of $\mathcal{NM}^{\{l\}}$ can be characterized as follows, for $1 \le i \ne j \le n$

1. $\eta_{ij}^{\{l\}} = 0$ if and only if $d_G(i,j) > 2^l$
2. $\eta_{ij}^{\{l\}} > 0$ if and only if $0 < d_G(i,j) \le 2^{l-1}$
3. $\eta_{ij}^{\{l\}} < 0$ if and only if $2^{l-1} < d_G(i,j) \le 2^l$

Proof Let us prove the theorem by induction on l. Basic step: When $l = 1$, by Definition 1 and Remark 1, we know that $\eta_{ij}^{\{1\}} \ne 0 \iff d_G(i,j) \le 2$. In particular, we have

$$\eta_{ij}^{\{1\}} = 0 \iff d_G(i,j) > 2 \tag{3}$$

$$\eta_{ij}^{\{1\}} > 0 \iff d_G(i,j) = 1 \tag{4}$$

$$\eta_{ij}^{\{1\}} < 0 \iff d_G(i,j) = 2 \tag{5}$$

Let us assume the theorem is true for any integer $p < k(G)$, that is,

$$\eta_{ij}^{\{p\}} = 0 \iff d_G(i,j) > 2^p \tag{6}$$

$$\eta_{ij}^{\{p\}} > 0 \iff 0 < d_G(i,j) \le 2^{p-1} \tag{7}$$

$$\eta_{ij}^{\{p\}} < 0 \iff 2^{p-1} < d_G(i,j) \le 2^p \tag{8}$$

Let us now prove for $p+1$:

1. $\eta_{ij}^{\{p+1\}} = 0 \iff \eta_{ij}^{\{l\}}$ is zero for every l, $1 \le l \le p+1$ (Which is shown in the proof of Theorem 3 (1)). Therefore, by induction, $d_G(i,j) > \underbrace{2 \cdot 2 \cdots 2}_{p+1 \text{ times}} = 2^{p+1}$.
2. $\eta_{ij}^{\{p+1\}} > 0 \overset{\text{By Def 1}}{\iff} i$ and j are adjacent in $G^{\{p+1\}}$, that is, $a_{ij}^{\{p+1\}} = 1 \overset{\text{By Def 2}}{\iff} \eta_{ij}^{\{p\}} \ne 0$. Hence, by Eqs. (7) and (8), we have $0 \le d_G(i,j) \le 2^p$.
3. Suppose $\eta_{ij}^{\{p+1\}} < 0$ then by Theorem 3 (1) we have $\eta_{ij}^{\{p\}} = 0$. Therefore, by induction, we have $d_G(i,j) > 2^p$. Further by Theorem 3 (2), we know that there exists a vertex x such that $\eta_{ix}^{\{p+1\}} > 0$ and $\eta_{xj}^{\{p+1\}} > 0$. Hence, by induction on the nonnegative entries, we have $d_G(i,x) \le 2^p$ and $d_G(x,j) \le 2^p$. Therefore $\eta_{ij}^{\{p+1\}} < 0 \implies 2^p < d_G(i,j) \le d_G(i,x) + d_G(x,j) \le 2^{p+1}$. Conversely, let $2^p < d_G(i,j) \le 2^{p+1}$. By induction on the nonnegative entries in the $\eta_{ij}^{\{p+1\}}$ namely using (1.) and (2.) above we have $d_G(i,j) > 2^p \implies \eta_{ij}^{\{p\}} = 0$, and $d_G(i,j) \le 2^{p+1} \implies \eta_{ij}^{\{p+2\}} > 0$. Since $\eta_{ij}^{\{p\}} = 0$ and $\eta_{ij}^{\{p+2\}} > 0$, by Theorem 3 (1), we have $\eta_{ij}^{\{p+1\}} < 0$.

Hence, the theorem is true for all integers l, $1 \le l \le k(G)$.

□

Proof of Theorem 5

Theorem 5 Let G be an undirected unweighted simple graph on n vertices and let $k(G)$ be the iteration number of G. If $\eta_{ij}^{\{l\}} = -1$, for $1 \le l \le k(G)$, then $d_G(i,j) = 2^l$.

Proof By the definition of $\mathcal{NM}$ matrix, $\eta_{ij}^{\{l\}} = -1$, implies that there exist exactly one common neighbour, say x, between i and j in $G^{\{l\}}$ and by Theorem 4, we have $2^{l-1} < d_G(i,j) = d_G(i,x) + d_G(x,j) \le 2^l$. Next, we claim that x is such a vertex that satisfies $d_G(i,x) = 2^{l-1}$ and $d_G(x,j) = 2^{l-1}$ and therefore, $d_G(i,j)$ becomes 2^l.

Claim: $d_G(i,x) = 2^{l-1}$ and $d_G(x,j) = 2^{l-1}$

Suppose f_1 is the vertex on the path connecting i and j such that $d_G(i,f_1) = 2^{l-1}$ and $d_G(f_1,j) \le 2^{l-1}$. By Theorem 4, we have $\eta_{if_1}^{\{l-1\}} < 0$ and $\eta_{f_1j}^{\{l-1\}} \ne 0 \implies \eta_{if_1}^{\{l\}} > 0$ and $\eta_{f_1j}^{\{l\}} > 0$ $\implies f_1$ is the common neighbour of i and j in $G^{\{l\}}$. Since x is the only common neighbour of i and j, we have $f_1 = x$, that is, $d_G(i,x) = 2^{l-1}$.

Similarly suppose f_2 is the vertex on the path connecting i and j, such that $d_G(f_2,j) = 2^{l-1}$ and $d_G(i,f_2) \le 2^{l-1}$, then by same argument as before, we get that $f_2 = x$. Hence, $d_G(x,j) = 2^{l-1}$.

Therefore, $d_G(i,j) = d_G(i,x) + d_G(x,j) = 2^{l-1} + 2^{l-1} = 2^l$.

□

Proof of Theorem 6

Theorem 6 A graph G is connected if and only if the corresponding matrix $\mathcal{NM}^{\{k\}}$ has no zero entries. In addition, the iteration number $k(G)$ of the graph $G \ne K_n$ is given by $k(G) = \lceil \log_2(diameter(G)) \rceil$. (For $G = K_n$, $k(G) = 1$ and $\mathcal{NM}^{\{1\}} = \mathcal{NM}(G) = -C(G)$)

Proof Let the number of nonzero entries in $\mathcal{NM}^{\{k\}}$ be denoted by z. Note that $z \le n^2$.

$$
\begin{aligned}
z = n^2 &\iff \mathcal{NM}^{\{k\}} \text{has no zero entries} \\
&\iff \eta_{ij}^{\{k\}} \ne 0, \forall i,j \in V(G) \\
&\iff 0 < d_G(i,j) \le 2^k, i \ne j, \forall i,j \in V(G) \text{ (Theorem 4)}
\end{aligned}
$$

Hence, G is connected and $diameter(G) \le 2^k$. Since $G \ne K_n$, we have $diameter(G) > 1$, implies that $k(G) \ge \log_2(diameter(G))$. Since the iteration number is defined to be the smallest possible integer satisfying the above condition, we obtain $k(G) = \lceil \log_2(diameter(G)) \rceil$.

□

Proof of Corollary 1

Corollary 1 A graph G is disconnected if and only if $z < n^2$. Further, the iteration number $k(G)$ of G is given by $k(G) = \lceil \log_2 S \rceil$, where S is the maximum over the diameter of components of G.

Proof Follows by contrapositive of above Theorem, since $z \le n^2$ and the observation that, if the (i,j)th entry of $\mathcal{NM}^{\{k\}}$ is zero then the (i,j)th entry of $\mathcal{NM}^{\{q\}}$ is zero, $\forall q \ge 1$. By definition, the iteration number $k(G)$ is the smallest number such that the number of nonzero entries in $\mathcal{NM}^{\{l\}}$ remains same for $l = k(G)$ and $l = k+1$. In particular, $k(G)$ will be obtained when the distance between every pair of vertices in every component has been accounted. Hence, $k(G) = \lceil \log_2 S \rceil$, where S is the maximum over the diameter of all the components of G. □

References

1. Abraham I, Bartal Y, Chan THH, Dhamdhere KD, Gupta A, Kleinberg J, Neiman O, Slivkins A (2005) Metric embeddings with relaxed guarantees. In: Proceedings of the 46th annual IEEE symposium on foundations of computer science, FOCS'05. IEEE Computer Society, Washington, DC, pp 83–100. https://doi.org/10.1109/SFCS.2005.51
2. Bellman E (1958) On a routing problem. Q Appl Math 16(1):87–90
3. Bondy J, Murty U (1976) Graph theory with applications. Macmillan, Berlin
4. Brunch T, Cornelissen K, Manthey B, Roglin H, Rosner C (2015) Smoothed analysis of the successive shortest path algorithm. SIAM J Comput 44(6):1798–1819
5. Chakaravarthy VT, Checconi F, Murali P, Petrini F, Sabharwal Y (2017) Scalable single source shortest path algorithms for massively parallel systems. IEEE Trans Parallel Distrib Syst 28(7):2031–2045
6. Coppersmith D, Winograd S (1990) Matrix multiplication via arithmetic progressions. J Symb Comput 9(3):251–280
7. Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms, 3rd edn. Prentice-Hall, New York
8. Dijkstra EW (1959) A note on two problems in connection with graphs. Numer Math 1(1):269–271
9. Dou Y, Zhu L, Wang HS (2012) Solving the fuzzy shortest path problem using multi-criteria decision method based on vague similarity measure. Appl Soft Comput 12:1621–1631
10. Duncan A (2004) Powers of the adjacency matrix and the walk matrix. Collect Univ Malta 9:4–11
11. Ebrahimnejad A, Tavana M, Alrezaamiri H (2016) A novel artificial bee colony algorithm for shortest path problems with fuzzy arc weights. Meas J Int Meas Confed (IMEKO) 93:48–56
12. Gao J, Zhao Q, Ren W, Swami A, Ramanathan R, Bar-Noy A (2015) Dynamic shortest path algorithms for hypergraphs. IEEE/ACM Trans Netw 23(6):1805–1817
13. Gao Y (2011) Shortest path problem with uncertain ARC lengths. Comput Math Appl 62:2591–2600

14. Goldberg AV, Harrelson C (2005) Computing the shortest path: a search meets graph theory. In: Proceedings of the 16th annual ACM-SIAM symposium on discrete algorithms, SODA'05. Society for Industrial and Applied Mathematics, Philadelphia, pp 156–165
15. Johnson DB (1977) Efficient algorithms for shortest paths in sparse networks. J ACM 24(1):1–13
16. Karunakaran S, Selvaganesh L (2019) An unique and novel graph matrix for efficient extraction of structural information of networks. CoRR arXiv:1903.05341, pp 1–10
17. Leskovec J (2012) Social circles: Facebook. https://snap.stanford.edu/data/egonets-~Facebook.html
18. Madkour A, Aref WG, Rehman FU, Rahman MA, Basalamah S (2017) A survey of shortest-path algorithms. arXiv:1705.02044 v1 [cs.DS], pp 1–26
19. Mulmuley K, Shah P (2001) A lower bound for the shortest path problem. J Comput Syst Sci 63(2):253–267. https://doi.org/10.1006/jcss.2001.1766
20. Nikolova E, Kelner JA, Brand M, Mitzenmacher M (2006) Stochastic shortest paths via quasi-convex maximization. In: Azar Y, Erlebach T (eds) Algorithms—ESA 2006. Springer, Berlin, pp 552–563
21. Potamias M, Bonchi F, Castillo C, Gionis A (2009) Fast shortest path distance estimation in large networks. In: Proceedings of the 18th ACM conference on information and knowledge management, CIKM'09. ACM, New York, pp 867–876. https://doi.org/10.1145/1645953.1646063
22. Roditty L, Shapira A (2011) All-pairs shortest paths with a sublinear additive error. ACM Trans Algorithms 7(4):45:1–45:12. https://doi.org/10.1145/2000807.2000813
23. Seidel R (1995) On the all-pairs-shortest-path problem in unweighted undirected graphs. J Comput Syst Sci 51(3):400–403
24. Sen S (2009) Approximating shortest paths in graphs. In: Proceedings of the 3rd international workshop on algorithms and computation, WALCOM'09. Springer, Berlin, pp 32–43. https://doi.org/10.1007/978-3-642-00202-1_3
25. Sheng Y, Gao Y (2016) Shortest path problem of uncertain random network. Comput Ind Eng 99:97–105
26. Sommer C (2014) Shortest-path queries in static networks. ACM Comput Surv 46(45):1–31
27. Thabet K, Al-Ghuribi S (2012) Matrix multiplication algorithms. Int J Comput Sci Netw Secur 12(2):74–79
28. West D (2000) Introduction to graph theory. Pearson, London
29. Xiao Y, Wu W, Pei J, Wang W, He Z (2009) Efficiently indexing shortest paths by exploiting symmetry in graphs. In: Proceedings of the 12th international conference on extending database technology: advances in database technology, EDBT'09. ACM, New York, pp 493–504. https://doi.org/10.1145/1516360.1516418
30. Zwick U (2001) Exact and approximate distances in graphs—a survey. In: Proceedings of the 9th annual European symposium on algorithms, ESA'01. Springer, London, pp 33–48. http://dl.acm.org/citation.cfm?id=647911.740642