DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

SOCIAL NETWORK ANALYSIS

CS 331

B.Tech, Sixth Semester

Submitted to:

Dr. Anupam Biswas

Guided by:

Ravi Kishore

A report on Localization of diffusion sources in complex networks with sparse observations

Group No. 2:

Ishika Raj- 2012050 *(Team Lead)*

Mudra Das – 2012011

Sadiqul Islam Choudhary - 2012048

Sumon Kumar Das - 2012069

Nirmalendu Sharma- 2012100

Saurav Kumar Boro - 2012072

INDIVIDUAL CONTRIBUTION

CODING IMPLEMENTATION

Sadiqul Islam Choudhary-2012048

Nirmalendu Sarma-2012100

TESTING AND ANALYSIS

Sumon Kumar Das-2012069

Ishika Raj-2012050

REPORT FORMATION

Mudra Das-2012011

Saurav Kumar Boro-2012072

ABSTRACT

In complex networks, it is often necessary to localise the sources of diffusion processes. This is particularly challenging when observations of the diffusion process are sparse or limited in some way. In this mini project, we explore the problem of localising diffusion sources in complex networks with sparse observations. Our approach is based on the backward diffusion-based method and integer programming. We evaluate the performance of our method on various network topologies and diffusion scenarios and show that it outperforms existing methods.

The backward diffusion-based method involves propagating information from the observation points back to the diffusion source. This is done by solving a diffusion equation in reverse time, starting from the observation points and working backwards towards the source. By doing so, the method is able to estimate the location of the diffusion source based on the information collected from the sparse observations.

Integer programming, on the other hand, is a mathematical optimization technique that can be used to solve combinatorial problems. In the context of diffusion source localization, integer programming is used to select the optimal set of observation points that can provide the most informative data for the backward diffusion-based method. By selecting the right set of observation points, the method can achieve a higher accuracy in localising the diffusion source with fewer observations.

Combining the backward diffusion-based method and integer programming allows for an efficient and effective approach to localise diffusion sources in complex networks with sparse observations.

INTRODUCTION

Diffusion in complex networks refers to the process of spreading information, substances, or phenomena through interconnected nodes or edges in a network. In many real-world applications, the diffusion process is affected by various factors that make it complex and challenging to understand and model accurately. Diffusion sources in complex networks refer to the nodes or edges in the network where the diffusion process originates. These sources can be, for example, pollutants, viruses, or hackers.

The localization of diffusion sources in complex networks with sparse observations is a problem that involves identifying the location of the source based on limited observations of the network. Sparse observations refer to the scenario where only a small subset of nodes or edges in the network are observed at each time step of the diffusion process. The limited and noisy nature of these observations makes the localization problem challenging.

To localise the diffusion source, various methods have been proposed, including graph-based methods, statistical methods, and optimization-based methods. These methods aim to estimate the source location based on the observed diffusion patterns in the network, taking into account the network topology and the properties of the diffusion process. Accurately localising diffusion sources in complex networks with sparse observations is crucial for effective response and prevention measures in various applications, including environmental monitoring, public health, and cybersecurity.

SOME BASIC CONCEPTS

Social network analysis [SNA] is the mapping and measuring of relationships and flows between people, groups, organisations, computers, URLs, and other connected information/knowledge entities. SNA provides both a visual and a mathematical analysis of human relationships

Rumour detection: When a rumour is detected in a network, one of the following two approaches can be adopted to minimise its spread:

1. **Direct Intervention**: In this approach, direct action is taken to stop the spread of the rumour. This can include measures like blocking or removing the source of the rumour, issuing a public statement to clarify the facts, or actively promoting an alternative narrative to counter the rumour. Direct intervention can be effective, but it can also be costly and may not always be feasible, especially if the source of the rumour is difficult to identify or if the rumour has already spread widely.
2. **Indirect Intervention:** In this approach, efforts are made to change the social dynamics of the network in order to reduce the likelihood that the rumour will spread. This can include measures like increasing the diversity of the network, promoting positive social norms, and encouraging individuals to be more critical of information they receive. Indirect intervention can be less intrusive and more sustainable than direct intervention, but it can also be slower and may require a longer-term investment in changing the social dynamics of the network.

Source Node: The node that is responsible for the spread of rumours in the network. Observer node: The node that is used as a point in the graph to observe the diffusion in the graph. It is used to locate the original source node.

Shortest path: The shortest path in graph theory is the path between two vertices (or nodes) in a graph that minimises the sum of the weights of its constituent edges.

Information diffusion: The dissemination of information across interconnected nodes or entities in a network is referred to as diffusion in a social network. The rate and intensity of diffusion are determined by network topology and network parameter initialization. Individual nodes in the diffusion process serve as a source of incentive for others

Distance Error: In network source localization, distance error refers to the difference between the shortest distance between an actual source node and the source node in the network detected by the algorithm. In case of multiple source nodes, distance error is the average of the shortest distances.

Sparse observations: In network analysis, sparse observations refer to situations where only a limited number of connections or interactions between nodes in a network are known or observed and thus the data may be incomplete or unavailable for certain nodes or edges.

MODELS AND METHODS

Propagation Model

A complex network is defined as a finite, undirected graphG = {V, E}, which consists of a set of nodes V and a set of edges E each edge denoting a connection through which information can be conveyed between the nodes. The topology of the network is assumed to be known and static during the diffusion process. The diffusion source set, $S = \{s_m\}_{m=1}^{M}$, is the set of the nodes that initiates the diffusion.

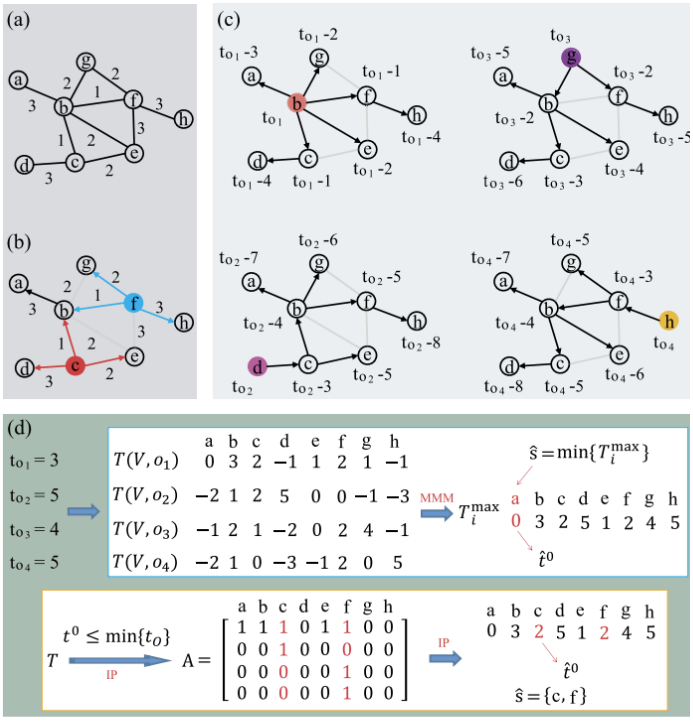


Fig.1. An example of the diffusion process with multisource

(a) is a weighted undirected graph *G*. The numbers on the edges are the corresponding time delays. The size of each node is in proportion to its degree.

(b) is the diffusion process initiated from sources $S = \{s_1, s_2\}$. The spreading paths from the two sources *s*1, *s*2 are highlighted by blue arrows and red arrows, respectively. The arrival time at observers O = $\{O_1,O_2,O_3,O_4\}$ is recorded.
(c) Illustration of the backward diffusion-based with four observers b, d, g, and h. The diffusion paths are highlighted by solid lines with arrows. The corresponding informed time tO for these observers are to1 , to2 , to3 and to4 . The inferred initial diffusion times are presented around each node via backward diffusion-based method.
(d) An example of the process of locating sources by MMM and IP.

Our model for characterising the spread of information over a complex network is implemented as follows:

1) In the network, at time t, all nodes are in two states: informed (the node has the information) or uninformed (the node does not have the information). Only the source nodes are in the informed state at the initial time t0.
2) The informed nodes forward the information to their uninformed neighbours along the shortest paths according to the corresponding delay time. For clarity, let us assume that node i receives the information for the first time and becomes informed at time ti. Then, i will transfer the information to all its neighbours, denoted as H(i), so that each uninformed neighbour node $j \in H(i)$ receives the information at time $t_i + \theta_{ij}$ where θ_{ij} is the transmission delay time of the edge (i, j), which follows a known joint distribution, e.g., Gaussian or uniform. Thus, for any node $i \in V$, the time t_i that i is informed is

$$t_i = t^0 + \min\{\Delta_{s_1,i}, \Delta_{s_2,i}, \cdots, \Delta_{s_M,i}\}, \quad (1)$$

where $\Delta_{s_m,i}$ is the shortest transmission time between s_m and i.

3) The process continues until all of the nodes in V have been informed.

Source Localisation

Let $O = \{o_k\}_{k=1}^{K}$ denote the set of K observers, whose informed times are known. Specifically, the observers' informed time is $\{t_{o_1}, t_{o_2}, \ldots, t_{o_K}\}$. Combining the network topology with delay time and using the backward propagation of each observer in O, we can infer the initial diffusion time of other nodes.

For each node i, if the transmission time $\Delta_{s_m,i}$ is the minimum of $\{\Delta_{s_1,i}, \Delta_{s_2,i}, \ldots \Delta_{s_M,i}\}$, then node i is informed by source sm. Let's define $i \in \Pi_{s_m}$, where Π_{s_m} is the diffusion set of source sm, shown by blue arrows and red arrows in Fig. 1(b). Thus, the original vertex set V can be decomposed into M diffusion set and one source node set, i.e., $V = \Pi_{s_1} + \Pi_{s_2} + \cdots + \Pi_{s_M} + S$.

So the relationship between the informed time and the diffusion set can be obtained. According to Eq. (1), for any node $i \in \Pi_{s_m}$, the informed time must be

$$t_i = t^0 + \min\{\Delta_{s_1,i}, \Delta_{s_2,i}, \ldots, \Delta_{s_M,i}\} = t^0 + \Delta_{s_m,i}. \quad (2)$$

Thus for any observer o_k, the inferred initial diffusion time of node s from the observer node should satisfy

$$T(s, o_k) = to_k - \Delta_{s,o_k} \leq t^0. \quad (3)$$

When s informs the observer o_k, then Δ_{s,o_k} must be the minimum in Δ_{s,o_k} set. As a result of Eq. (3), the true initial diffusion time t^0 is the maximum of $T(s, o_k)$, denoted as T_s^{max}. In order to identify all sources,

we should infer the initial diffusion time of all nodes from backward diffusion-based method, namely T ($i \subset V$, $o_k \subset O$). From this we can readily obtain T_i^{max} from all observers O. We select the nodes with the minimum value of T_i^{max} as sources, which we refer to as the maximum-minimum method (MMM).

To demonstrate the aforementioned method, we will provide a specific example. Let us consider the edge (i, j) as a random integer chosen from a uniform distribution U(1, 5), as shown in Fig. 1(a). For this example, let us assume that nodes c and f are the sources, as depicted in Fig. 1(b). We also assume that the initial diffusion times collected from observers b, d, g, and h are tb = 3, td = 5, tg = 4, and th = 5. Using the backward diffusion method from these observers, we can estimate the initial diffusion time

$$T(V,O) = \begin{bmatrix} & a & b & c & d & e & f & g & h \\ b & 0 & 3 & 2 & -1 & 1 & 2 & 1 & -1 \\ d & -2 & 1 & 2 & 5 & 0 & 0 & -1 & -3 \\ g & -1 & 2 & 1 & -2 & 0 & 2 & 4 & -1 \\ h & -2 & 1 & 0 & -3 & -1 & 2 & 0 & 5 \end{bmatrix}$$

Thus T_i^{max} for each node $i \subset V$ from the observers is

$$\begin{bmatrix} a & b & c & d & e & f & g & h \\ 0 & 3 & 2 & 5 & 1 & 2 & 4 & 5 \end{bmatrix}$$

Algorithm of Integer Programming

Two methods are proposed for locating sources in a network. The first method uses the MMM approach, which involves identifying the nodes

with the minimum value of T_i^{max} and assuming that the source is located at one of these nodes, with an inferred initial time of t^0 = 0. However, this approach may not accurately identify the true sources c and f. Therefore, a second method is developed using 0-1 integer programming.

The potential sources are nodes that satisfy $T_i^{max} \leq \min\{t_0\}$, denoted by Ω. The set {a, b, c, e, f} is identified as potential sources in Ω. By setting all T_i^{max} in Ω to 1 and the others to 0, a matrix T(V, O) can be constructed as

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

The algorithm provided above is a specific example of using 0-1 integer programming to locate sources based on a set of observation times. Here is how the algorithm works:

The 0-1 integer programming algorithm is a general approach for solving optimization problems where the decision variables can take on binary values (either 0 or 1). The goal is to find the optimal combination of binary values that satisfy certain constraints while minimising or maximising an objective function.

In the context of source localization based on diffusion time, the goal is to determine the locations of the sources in a network such that the observed diffusion times from each source to a set of observation points are explained. This problem can be formulated as a 0-1 integer program,

where the decision variables are binary values indicating the presence or absence of a source at each node in the network.

In the example provided, the first step is to identify the set of potential source nodes based on the minimum observation time. The set of potential source nodes, denoted by Ω , is the set of nodes where the maximum diffusion time to any observation point is less than or equal to the minimum observation time. In this case, the set {a, b, c, e, f} is identified as potential sources.

The next step is to define the binary variable vector X, where each element x_i takes on the value of either 0 or 1, indicating the presence or absence of a source at a particular node. In this example, the length of the vector is 8, since there are 8 nodes in the network.

The objective function of the 0-1 integer program is to minimise the number of sources needed to explain the set of observation times. This can be done by minimising the sum of the binary variables in the vector X.

The main constraint in this problem is that each observation point must receive a diffusion signal from at least one source. This constraint can be expressed as a set of linear inequalities, where each observation point is associated with a set of source nodes that can potentially explain its diffusion time. In this example, the matrix T(V,O) is translated into the constraint matrix A, where each row of the matrix represents an observation point and each column represents a potential source node. The element A_{ij} of the matrix is 1 if node j is a potential source for observation point i, and 0 otherwise.

Once the 0-1 integer program is formulated with the objective function and constraints, it can be solved using a specialised solver, which will

give the optimal solution to the problem. In this example, solving the 0-1 integer program with the matrix A will give the optimal combination of binary variables that satisfy the constraints and minimise the objective function.

Finally, the feasibility of the solution can be checked by verifying that the estimated sources can explain all the observation times. If the solution is feasible, the estimated source locations can be refined using additional techniques, such as triangulation or signal strength analysis.

Overall, the 0-1 integer programming algorithm is a general approach for solving optimization problems where the decision variables can take on binary values, and it can be applied to the problem of source localization based on diffusion time by formulating the problem as a 0-1 integer program with appropriate objective function and constraints.

Floyd-Warshall Algorithm

We have also implemented the Floyd-Warshall algorithm to compute the shortest path between every pair of nodes in a network. The function takes a 2D list graph as input, where graph[i][j] represents the weight of the edge connecting node i and node j. The function returns a 2D list dist where dist[i][j] represents the shortest path between node i and node j.

Finding Distance Error

Result and Source are lists of indices, where Result contains the indices of the nodes that are identified as potential source locations by the algorithm, and Source contains the indices of the true source locations in the network. dp is a 2D array containing the pairwise distances between nodes in the network. Firstly, we use a function that sorts the Result and Source lists, and then computes the minimum distance

between each true source location and each estimated source location using the pairwise distances stored in dp. The distance is summed over all pairs of true and estimated source locations, and then divided by the number of pairs that were considered. One variable is set to the minimum of the lengths of Result and Source, which ensures that the distance calculation is only performed for pairs of nodes that have corresponding indices in both lists. If there are more estimated source locations than true source locations, the function only considers the true source locations, and vice versa.

Overall, this function can be used to evaluate the accuracy of a source localization algorithm by comparing its estimated source locations to the true source locations, and calculating the average distance error between them.

ANALYSIS OF THE ALGORITHM

To measure the performance, validity and efficiency of source localization, we will use the f1 score, which is commonly used in information retrieval for assessing search, document classification, and query classification performance. The f1 score is calculated as the harmonic mean of Precision and Recall, where larger values indicate better performance. Precision and Recall are computed using the formulas given below:

Precision = TP/(TP + FP) and

Recall = TP/(TP + FN),

where TP, TN, FP, and FN represent true positives, true negatives, false positives, and false negatives, respectively.

Therefore, f1 = 2(Precision . Recall)/(Precision + Recall)

The range of the f1 score is 0 to 1, where larger f1 scores correspond to better performance.

CODE AND IMPLEMENTATION

1. Importing the libraries

Here, we import the libraries required for the implementation of the algorithm.

```
import networkx as nx
import csv
import matplotlib.pyplot as plt
import numpy as np
import random
import time
```

2. Propagating the rumour through the network

Given a source node, we've calculated the time at which the rumour reaches each node in the network. The function propagate takes four arguments: source (the index of the source node), arr (a list to store the propagation time for each node), dp (a 2D list representing the shortest path between every pair of nodes in the network), and t0 (the time at which the rumour starts spreading from the source node).

```
def propagate(source,arr,dp,t0):
    sourceLen=len(source)
    V=len(dp)
    for i in range(V):
        arr[i]=INF;
    for i in range(sourceLen):
        arr[source[i]]=t0
    for i in range(sourceLen):
        for j in range(V):
            arr[j]=min(arr[j],t0+dp[source[i]][j])
    return arr
```

3. Calculating the shortest path using Floyd-Warshall Algorithm

The floydWarshall function takes the adjacency matrix of the network as input and returns the shortest path matrix dp between all pairs of nodes in the network.

```
def floydWarshall(graph):
    V=len(graph)
    dist = list(map(lambda i: list(map(lambda j: j, i)), graph))
    for k in range(V):
        # pick all vertices as source one by one
        for i in range(V):
            # Pick all vertices as destination for the above picked source
            for j in range(V):
                # If vertex k is on the shortest path from
                # i to j, then update the value of dist[i][j]
                dist[i][j] = min(dist[i][j],
                                 dist[i][k] + dist[k][j]
                                 )

    return dist;
```

4. Finding Time Stamp

Here we've calculated the time at which each node in the network receives the rumour. The function FindingTimeStamp takes four arguments: observer, TimeStamp (a 2D list to store the time at which each node receives the rumour), dp, and arr.

```
def FindingTimeStamp(observer,TimeStamp,dp,arr):
    for i in range(len(observer)):
        for j in range(len(dp)):
            TimeStamp[i][j]=arr[observer[i]]-dp[observer[i]][j]
    return TimeStamp
```

5. Finding the Source Node using integer programming

Now to find the source node of the rumour given the propagation time of the rumour to each node in the network, we have implemented a simple

algorithm which finds the minimum subset of columns that covers all the rows with a value of 1 in the TimeStamp matrix.

```
def FindSourceNode(TimeStamp):
    print(" To find the minimum subset of columns that covers all the rows
with a value of 1. This minimum subset is our answer")
    rows = len(TimeStamp)
    cols = len(TimeStamp[0])
    covered = set()
    result = []
    while len(covered) < rows:
        max_col = None
        max_count = 0
        for j in range(cols):
            if j not in result:
                count = 0
                for i in range(rows):
                    if TimeStamp[i][j] == 1 and i not in covered:
                        count += 1
                if count > max_count:
                    max_col = j
                    max_count = count
        if max_col is not None:
            result.append(max_col+1)
            for i in range(rows):
                if TimeStamp[i][max_col] == 1:
                    covered.add(i)

    result.sort()
    print(result)
    return result
```

5. Calculating the F1, True Positive, False Positive and False Negative for Integer Programming

Here we've calculated the F1 score,True Positive, False Positive and False Negative for the source node identified by the algorithm we 've used. The function takes two arguments: Result (the list of source nodes identified by the algorithm) and Source (the list of actual source nodes).

```
def CalculateScore(Result,Source):
    print("F1 score and other measures for Our Algo are ")
    Result.sort()
    Source.sort()
    n1=len(Result)
    n2=len(Source)
    i=0
    j=0
    TP=0
    FP=0
    FN=0
    while i<n1 and j<n2:
        if(Result[i]==Source[j]):
            TP+=1
            i+=1
            j+=1
        elif(Result[i]<source[j]):
            FP+=1
            i+=1
        else:
            FN+=1
            j+=1
    if(i!=n1):
        FP=FP+(n1-i)
    if(j!=n2):
        FN=FN+(n2-j)
    print("True Positive are:")
    print(TP)
    print("False Positive are:")
    print(FP)
    print("False Negative are:")
    print(FN)
    Precision=TP/(TP+FP)
    Recall=TP/(TP+FN)
    if((Precision+Recall)==0):
```

```
        print("F1 Score Cannot be calculated as denomitor is 0")
    else:
        f1=2*(Precision*Recall)/(Precision+Recall)
        print("The F1 Score is ")
        print(f1)
    return
```

6. Calculating Distance Error

The DistanceError function takes as input two lists of indices (Result and Source) and dp. It calculates the average distance error between the estimated source locations in Result and the true source locations in Source, by computing the minimum distance between each pair of true and estimated source locations using the distances stored in dp. The resulting distance error is the sum of these minimum distances, divided by the number of pairs that were considered.

```
def DistanceError(Result,Source,dp):
    Result.sort()
    Source.sort()
    n1=len(Result)
    n2=len(Source)
    distance=0
    lim=min(n1,n2)
    if n2<n1:

        for i in range(n2):
            temp=INF
            for j in range(n1):
                temp=min(dp[Source[i]][Result[j]],temp)
            if(temp==INF):
                lim-=1
            else:
                distance+=temp
    else:
        for i in range(n1):
            temp=INF
            for j in range(n2):
                temp=min(dp[source[j]][Result[i]],temp)
            if(temp==INF):
```

```
                lin-=1
            else:
                distance+=temp

    return distance/lim
```

7. The Main function

It takes an adjacency matrix of a graph, propagates the infection from randomly selected source nodes, finds the timestamp of all nodes using reverse diffusion, calculates the maximum timestamp of each node with respect to all observers, and finally applies integer programming to determine the source nodes. It also prints the distance error and time taken for the algorithm to run.

```
if __name__ == "__main__":
starttime=time.time()
g=nx.Graph()
with open("dolphins.csv", mode ='r')as file: #input function of graphs
    csvFile = csv.DictReader(file)
    for line in csvFile:
       g.add_edge(int(line['Source'])-1,int(line['Target'])-1)

#Function call
#graph to 2d array A is a 2d Array of graph g
A = nx.adjacency_matrix(g).todense()

#node is the number of nodes in the graph
node=len(A)
print("This graph contain nodes:-")
print(node)
for i in range(node):
    for j in range(node):
        if(A[i][j]==0):
            A[i][j]=INF
        if(i==j):
            A[i][j]=0;

#dp contain shortest path length from all nodes to all other nodes
dp=floydWarshall(A)
```

```
#arr will contain the timestamp of all the node when the node gets infected
arr=[0]*node

sourcelen=10
observerlen=4
#source array containing all the source nodes
source=[0]*sourcelen
#observer array containing all the observer nodes
observer=[0]*observerlen
for i in range(sourcelen):
    source[i]=random.randint(0,node-1)

for i in range(observerlen):
    observer[i]=random.randint(0,node-1)

source.sort()
observer.sort()
print("Sources are :")
print(source)
print("Observer are :")
print(observer)

#t0 is the initial timestamp of sources
t0=2;
arr=propagate(source,arr,dp,t0)

#Finding the timestamp of nodes using reverse diffusion
TimeStamp=[[0]*node for _ in range(len(observer))]
TimeStamp=FindingTimeStamp(observer,TimeStamp,dp,arr)

#Now we will calculate Maximum Timestamp of all node with respect to all
observer
MaximumTimeStamp=[-INF]*node
MinTimeStamp=INF;
for i in range(len(observer)):
    for j in range(node):
        MaximumTimeStamp[j]=max(MaximumTimeStamp[j],TimeStamp[i][j])

for i in range(node):
```

```
    MinTimeStamp=min(MinTimeStamp,MaximumTimeStamp[i]);
print("According MMM the source node are are the nodes with maximum Maximum
Timestamp")
ResultMMM=[]
for i in range(node):
    if(MaximumTimeStamp[i]==MinTimeStamp):
        ResultMMM.append(i)
print(ResultMMM)
# #we are dividing the node in two sets one which may contain source node and
another which will not
check=[0]*node
minObserverTime=INF
for i in range(len(observer)):
    minObserverTime=min(minObserverTime,arr[observer[i]])

for i in range(node):
    if(MaximumTimeStamp[i]<=minObserverTime):
        check[i]=1
print("After applying the Integer programming the Timestamp matrix will be
translated to")
for i in range(len(observer)):
    for j in range(node):
        if(check[j]==0):
            TimeStamp[i][j]=0
        else:
            if(TimeStamp[i][j]==MaximumTimeStamp[j]):
                TimeStamp[i][j]=1;
            else:
                TimeStamp[i][j]=0;
result=FindSourceNode(TimeStamp)
CalculateScore(result,source)
endtime=time.time()
print("The Distance Error for this input is
",DistanceError(result,source,dp))
print("Time taken for this is =", endtime-starttime,"seconds")
```

EXPERIMENT AND ANALYSIS

DESCRIPTION OF DATASETS

We have used ten datasets:

- Football
- Dolphin
- Karate
- BA_500_1_giant_edge_list
- BA_500_2_giant_edge_list
- ER_500_1_giant_edge_list
- ER_500_4_giant_edge_list
- BA_1000_1_giant_edge_list
- BA_1000_2_giant_edge_list
- BA_1000_5_giant_edge_list

First let us see the basic attributes of the datasets used for analysis of this algorithm

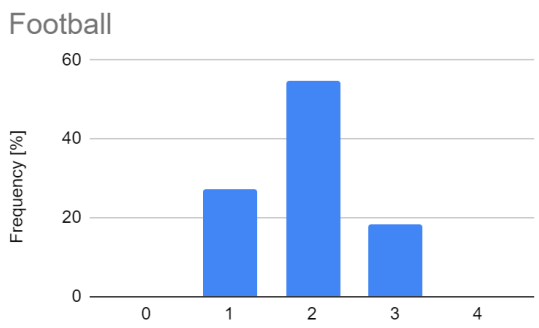
Sl. No.	Name of Datasets	Nodes
1	Football	115
2	Dolphin	62
3	Karate	34
4	BA_500_1_giant_edge_list	486
5	BA_500_2_giant_edge_list	519
6	ER_500_1_giant_edge_lis	500

7	ER_500_4_giant_edge_lis	500
8	BA_1000_1_giant_edge_list	1000
9	BA_1000_2_giant_edge_list	1000
10	BA_1000_5_giant_edge_list	1000

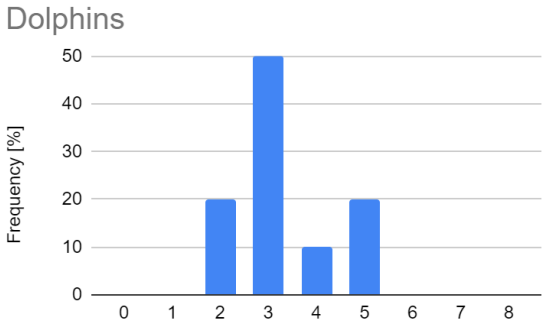
We will be now comparing the above datasets with various variables for our algorithm.

1. Frequency vs Distance Error

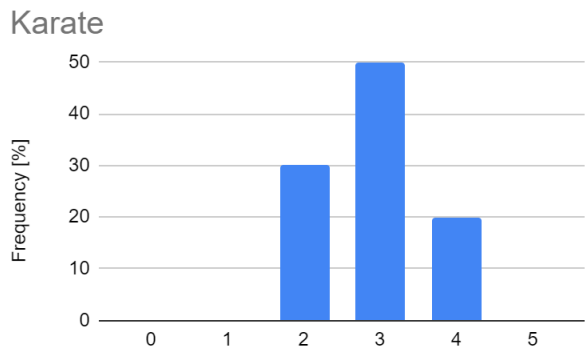
i)



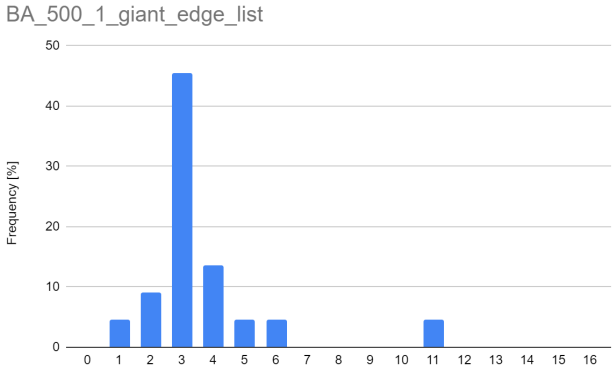
ii)



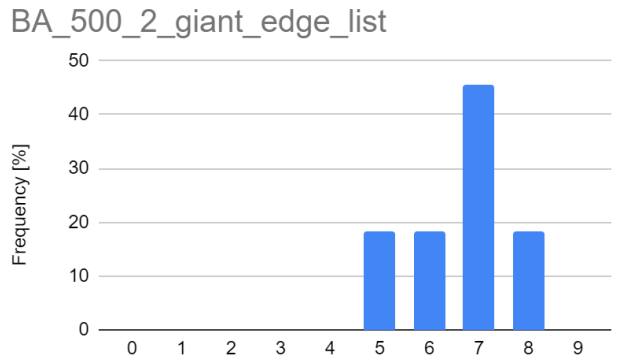
iii)



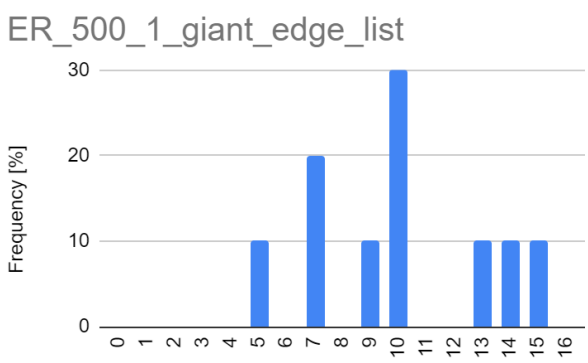
iv)



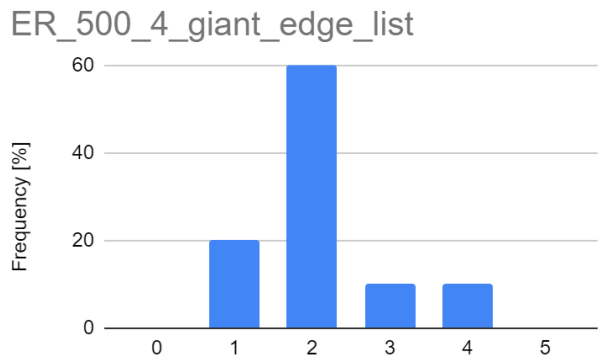
v)



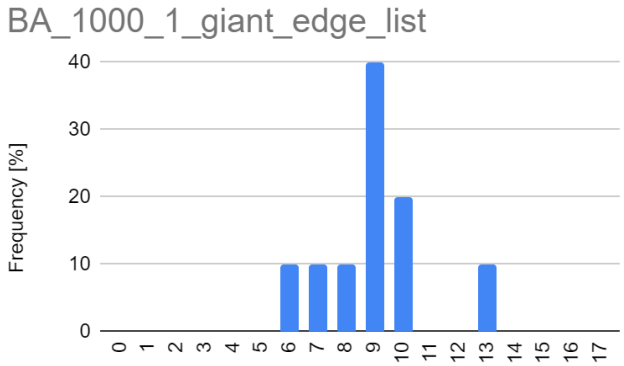
vi)



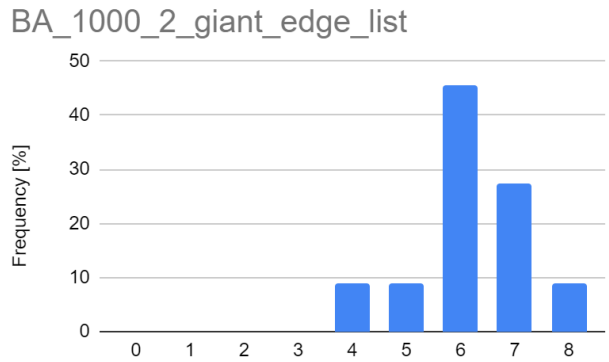
vii)



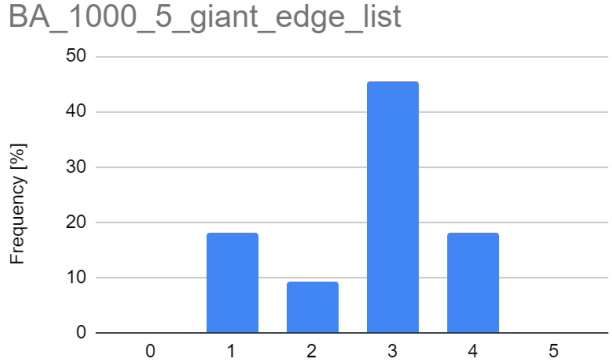
viii)



ix)

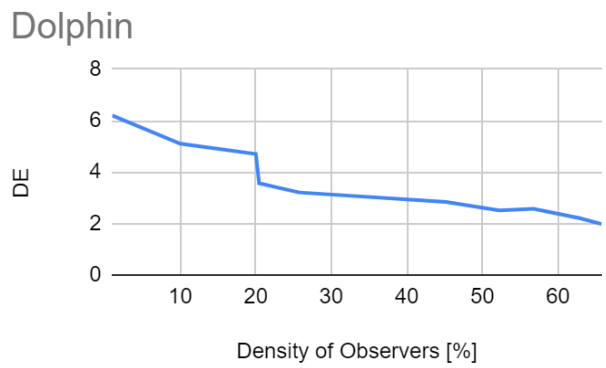


x)

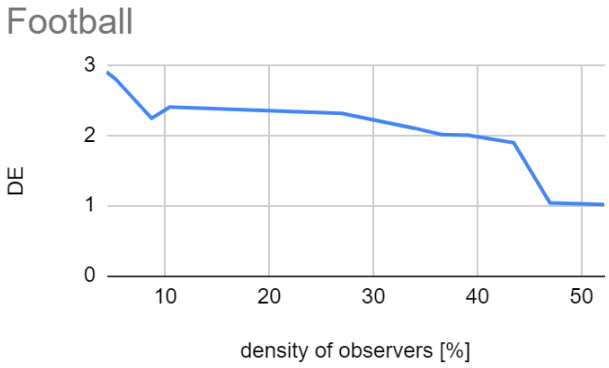


2. Distance Error vs Density of Observers

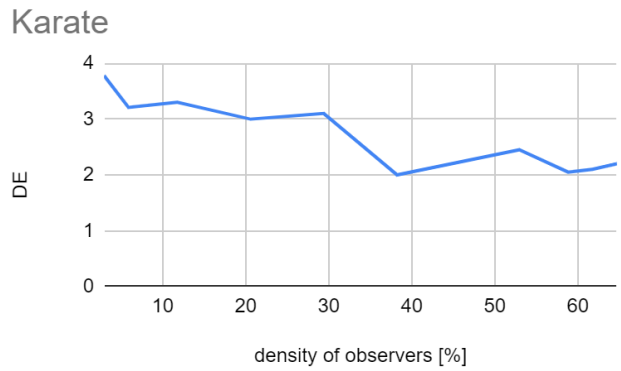
i)



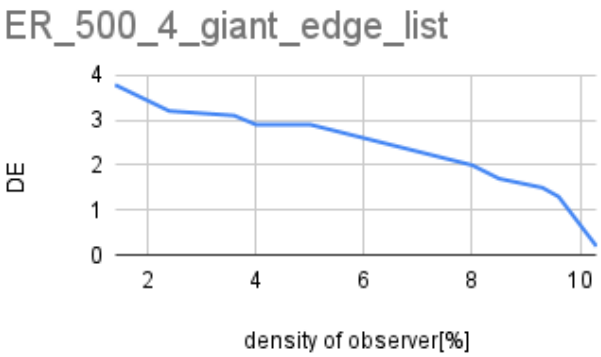
ii)



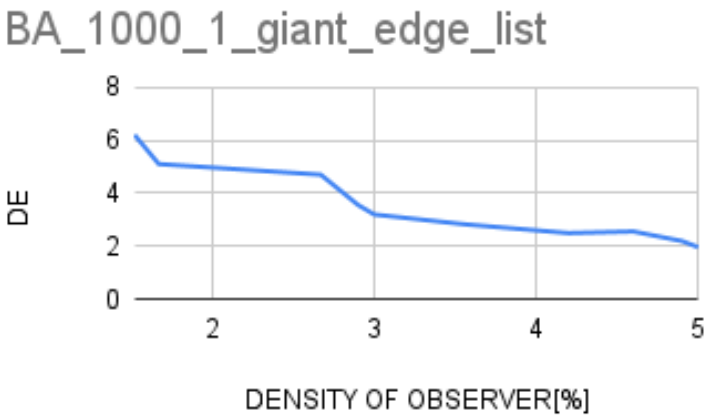
iii)



iv)

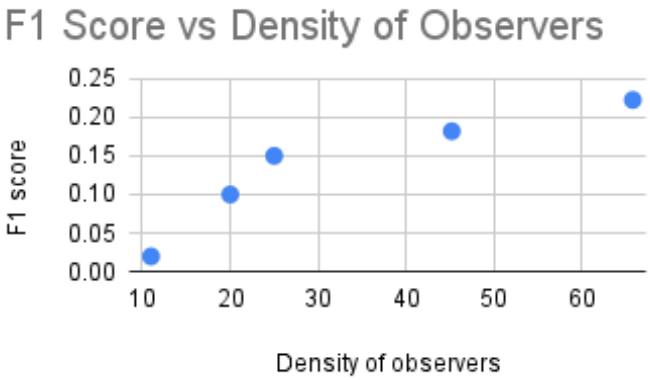


v)

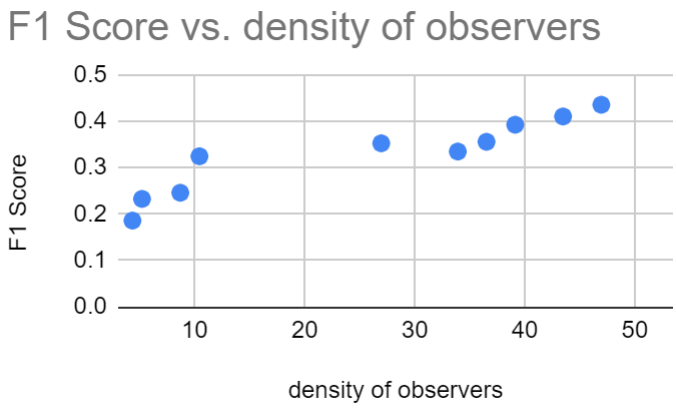


3. F1 Score vs Density of Observers

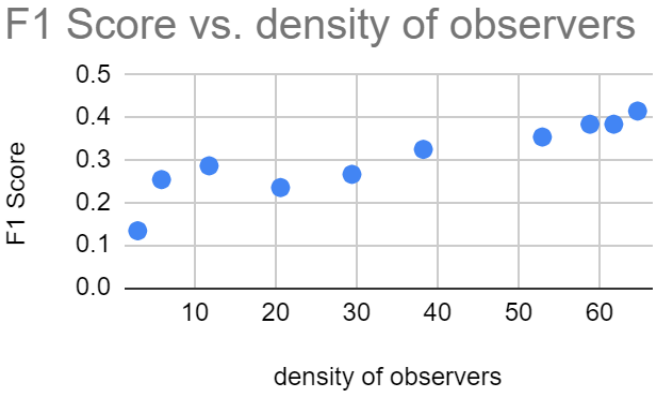
i) Dolphin



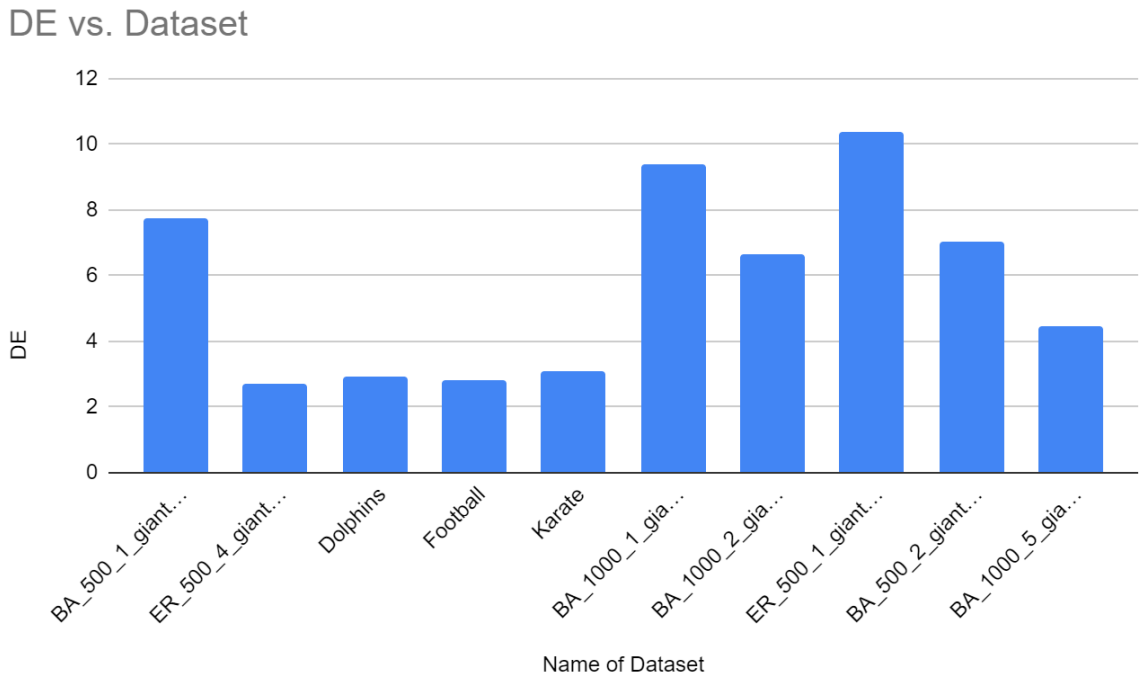
ii) Football



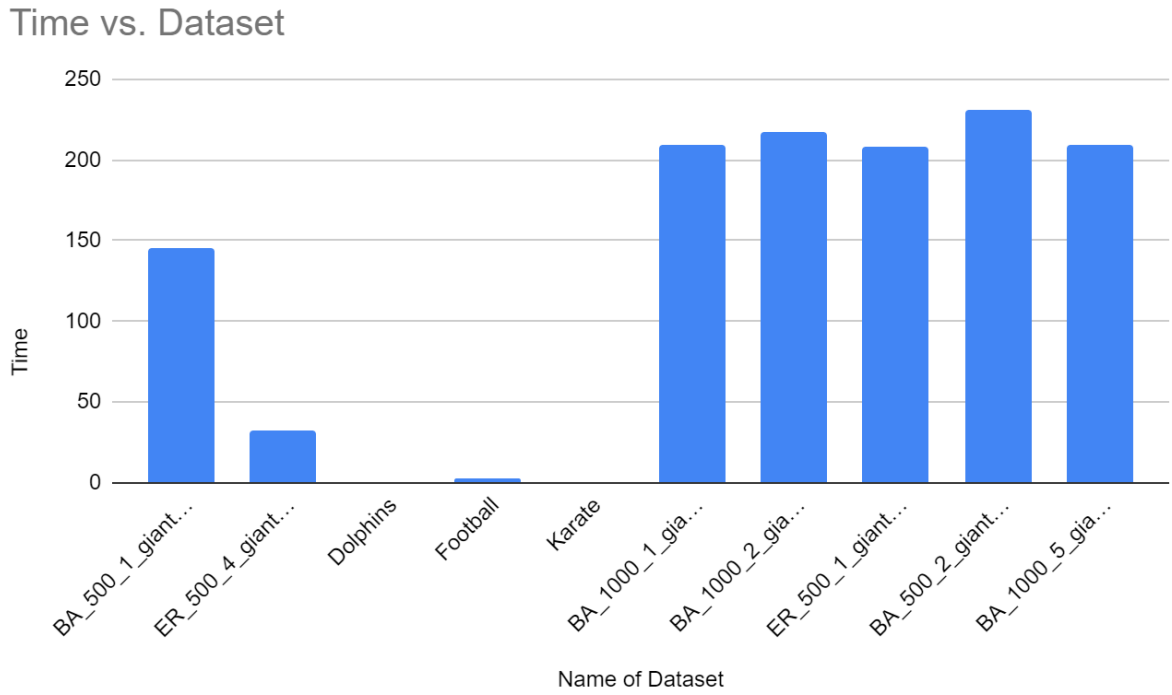
iii) Karate



4. Distance Error vs Dataset



4. Time vs Dataset



DRAWBACK

If there exists a node in the graph such that the difference between receiving time and shortest distance between that node and the set of observers are constant. Then our algorithm will show that node as the source node irrespective of the actual source node. To get a better understanding of this drawback, let us consider an example where node i satisfies the above condition.

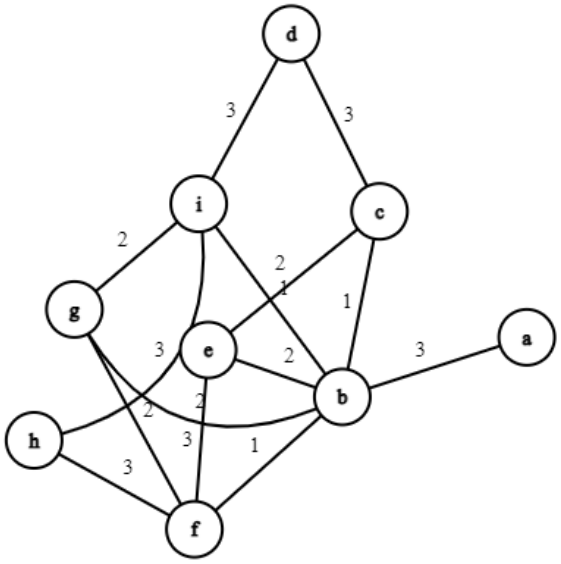
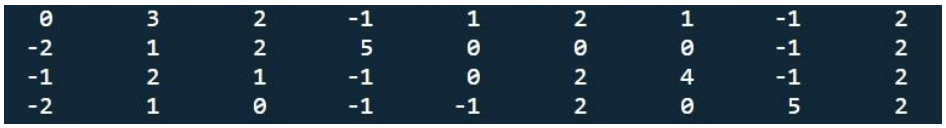


Fig 2.

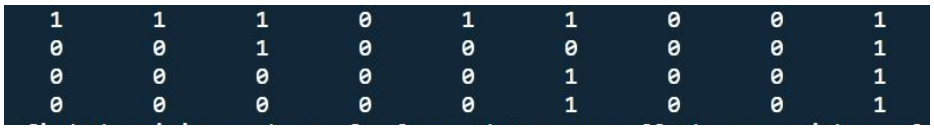
In Fig 2., Source node ={c,f}, Observer = {b,g,d,h}, Initial Time = t0 =2
After propagating, the receiving time of all nodes will be {6, 3, 2, 5, 4, 2, 4, 5, 4} respectively.
For node i, the shortest distance between observers {b, g, d, h} and node i are {1, 3, 2, 3} respectively and the difference between receiving time of observers and shortest distance are {2, 2, 2, 2} which is constant.

Now after backtracking, the timestamp of all nodes with respect to observer set will be



0	3	2	-1	1	2	1	-1	2
-2	1	2	5	0	0	0	-1	2
-1	2	1	-1	0	2	4	-1	2
-2	1	0	-1	-1	2	0	5	2

After applying Integer Programming, the timestamp matrix will be translated to



1	1	1	0	1	1	0	0	1
0	0	1	0	0	0	0	0	1
0	0	0	0	0	1	0	0	1
0	0	0	0	0	1	0	0	1

And the output will contain node i as source node since column of i has all entries as 1 in the above matrix.

But our actual sources are c and f so we observed that such nodes exist in the graph, the algorithm assumes that node is the source.

CONCLUSION

We have come across two methods for locating diffusion sources in complex networks : Maximum–minimum method(MMM) and Integer Programming.

The first algorithm infers initial diffusion time of information in a network of nodes by tracing information spread backwards from observers to source nodes using delay times and network topology. The method selects the minimum value among the maximums Timestamp as the most likely time when the information was initially spread from the source node. One disadvantage of the Maximum-Minimum Method (MMM) is that it may not accurately identify the true sources in all cases. This is because the method only considers the minimum value of maximum Timestamp as the criterion for identifying potential sources. It is possible that some nodes may have similar Timestamp values, and therefore the MMM method may not be able to distinguish between them as potential sources. Additionally, the MMM method assumes that the inferred initial time of the sources is 0, which may not always be the case in real-world scenarios. As for the second algorithm which is integer programming, its goal is to determine the locations of sources in a network using binary variables that minimise the number of sources needed to explain observed diffusion times while satisfying constraints.

Our approach involves merging backward diffusion-based techniques with IP to determine the locations of sources and the initial diffusion time, even with a limited number of observers. This method relies on obtaining delay time and informed times from partial observers.

Our method first identifies the set of nodes which can be the source nodes which are having less than or equal to the minimum receiving

time of all the observers to get a better result. It then finds the minimum no of nodes which can cover all the nodes in the graph using Integer programming.

We found that our technique works best on large networks and performs better than MMM when only a few observers are available, particularly in heterogeneous networks. Additionally, our method is not affected by noise and is resilient against different observer selection strategies, making it possible to select observers randomly. We confirmed these findings through both simulated and real-world network data.

- The model performance decreases with the number of sources.
- For a small no, the performance of IP is much better than that of MMM.
- Both methods perform well in homogeneous Networks.
- a larger network size is beneficial for source localization.
- Our method will perform much better than MMM, in particular, for heterogeneous networks.
- It is easier to locate sources in homogeneous networks for both methods.

Both of the Algorithms can be further modified to improve the accuracy and less execution time

Following is the Google Collab Link to the code we coded for the above two algorithms:

https://colab.research.google.com/drive/1uo_6WfLBQ4FXXbpUJjClkxHixFiHQy8x?usp=sharing

REFERENCES

1. (Original Paper) Zhao-Long Hu, Zhesi Shen, Chang-Bing Tang, Bin-Bin Xie, Jian-Feng Lu, Localization of diffusion sources in complex networks with sparse observations.
2. L. Fu, et al., Multi-source localization on complex networks with limited observers, Europhys. Lett. 113 (2016) 18006.