Comparative Analysis of Graph Algorithms on Various Case Studies

Submitted To-:
Dr. Animesh Chaturvedi
Assistant Professor: IIIT Dharwad
Young Researcher: Heidelberg Laureate Forum
Postdoc: King's College London & The Alan Turing Institute
PhD: IIT Indore
MTech: IIITDM Jabalpur

Team Members-:

- Arpit Nayak 22BDS006
- Ashutosh Wani 22BDS008
- Rajput Ajay 22BDS049
- Viraj Surana 22BDS064
- Bhabad Ganesh 22BDS067

Content-:

Abstract -:

This project aims to analyze the performance of four different graph algorithms on two case studies. The first case study focuses on the analysis of example graphs, treating them as undirected graphs. The second case study involves analyzing a dataset containing diverse graph structures. The chosen algorithms are Prim's, Kruskal's, Dijkstra's, and Bellman-Ford's, while the properties considered are mean, median, minimum, and maximum. The project utilizes time-series plots to generate meaningful observations and findings. The results are presented through a write-up report, PowerPoint slides, and a video presentation.

Dataset Description-:

Considering the dataset of "Resistance game", each vertices in the network represents a participant in a game, and it is characterized by a binary attribute indicating their role as either a deceiver or a non-deceiver. The edges in the network represent the interactions between pairs of participants during the game, with each interaction corresponding to a specific second.

Algorithm Description-:

The algorithms used to process the data of the dataset are Prim's, Kruskal's, Dijkstra's and Bellman-ford's. Prim's algorithm selects the shortest edge to connect unvisited vertices, creating a minimum spanning tree, Kruskal's algorithm adds edges in ascending order of weight, avoiding cycles to form a minimum spanning tree, Dijkstra's algorithm finds the shortest path from a source to all vertices by selecting the vertex with the smallest distance and updating its neighbors and Bellman-Ford's algorithm computes the shortest path from a source to all vertices, allowing negative edge weights, by iteratively relaxing edges and updating distances. The useful idea in the reference is the proposal of using Face-to-Face Dynamic Interaction Networks (FFDINs) to model interpersonal interactions within a group of people in order to detect group deception in video conversations. The novel idea in the reference is the introduction of Negative Dynamic Interaction Networks (NDINs) to capture the concept of missing interactions, and the development of the Deception Rank algorithm to detect deceivers from NDINs extracted from short videos, outperforming existing methods in identifying deception.

Experimentations, Results, Observations, Findings, Inferences-:

Experimentations: The authors compared the performance of their proposed method, DeceptionRank, with state-of-the-art vision and graph embedding baselines. They conducted experiments on a dataset of 26 games, augmenting it by segmenting long videos into smaller ones, resulting in 2781 data points. The dataset was split into training and test sets using 5-fold cross-validation to prevent leakage of ground-truth labels. Both vision-based and graph embedding baselines were evaluated, along with an ensemble baseline that combined the strengths of all baselines.

Results: DeceptionRank outperformed all baseline methods by at least 20.9% in detecting deceivers from 1-minute clips of videos. The proposed method achieved an AUROC of 0.753, with a 95% confidence interval ranging from 0.721 to 0.789. The ensemble baseline performed the best among the baselines, but it was still surpassed by DeceptionRank.

Findings: DeceptionRank demonstrated superior performance in detecting deceivers compared to both vision-based and graph embedding baselines. It also remained robust across various segment lengths, while the baseline methods showed diminished performance with shorter segments.

Observations: The authors noticed that DeceptionRank's performance was better in games where deceivers won (DL games) compared to games where non-deceivers won (DW games). This observation was attributed to the significantly different behaviors of deceivers and non-deceivers in DL games, making it easier for the algorithm to distinguish between them.

Inferences: The results indicated that DeceptionRank is a powerful approach for detecting deceivers in group interactions, even in short video segments. The proposed Negative Dynamic Interaction Network and DeceptionRank algorithm can be valuable for applications in web and social media platforms, enhancing safety and integrity in video-based communications. Future work could explore applying these methods to other datasets and settings to further validate their effectiveness in detecting deception

Pros:

1) Novel approach: Introduces innovative Face-to-Face Dynamic Interaction Network (FFDIN) and Negative Dynamic Interaction Network (NDIN) concepts for group deception analysis.
2) Superior performance: Outperforms state-of-the-art baselines by at least 20.9%, validating the effectiveness of DeceptionRank algorithm.
3) Real-world relevance: Addresses group deception in video-based conversations, relevant for online communication and social media safety.
4) Generalizability: Applicable to various group interaction settings beyond the specific game used in the study.

Cons:

1) Limited dataset: Relies on a single game dataset, limiting generalization to other contexts.
2) Dataset creation effort: Time-consuming to create a scarce dataset with ground truth of group deception.
3) Lack of real-world evaluation: Future work needed to test the method on web and social media data.
4) Dependency on interaction quality: Accuracy could be affected by video and audio quality.

Citation of the Reference-:

S. Kumar, C.Bai, V.S. Subrahmanian, J. Leskovec. Deception Detection in Group Video Conversations using Dynamic Interaction Networks. 15th International AAAI Conference on Web and Social Media (ICWSM), 2021. C.Bai, S. Kumar, J. Leskovec, M. Metzger, J.F. Nunamaker, V.S. Subrahmanian. Predicting Visual Focus of Attention in Multi-person Discussion Videos. International Joint Conference on Artificial Intelligence (IJCAI), 2019.

Algorithm-:

1. Prim's Algorithm- https://github.com/viraj-surana/ds_project.git, Purpose-To find a minimum spanning tree (MST) in a connected weighted undirected graph

2. Bellmanford Algorithm- https://github.com/viraj-surana/ds_project.git, Purpose-The purpose of the Bellman-Ford algorithm is to find the shortest paths from a source vertex to all other vertices in a weighted directed graph

3. Kruskal Algorithm- https://github.com/viraj-surana/ds_project.git. Purpose-To find a minimum spanning tree (MST) in a connected weighted undirected graph

4. Dijkstra's algorithm(DijkstraSP.java)-https://github.com/viraj-surana/ds_project.git, Purpose- to find the shortest path between a starting vertex and all other vertices in a weighted graph. It is applicable to graphs with non-negative edge weights.

Results of Dataset-:

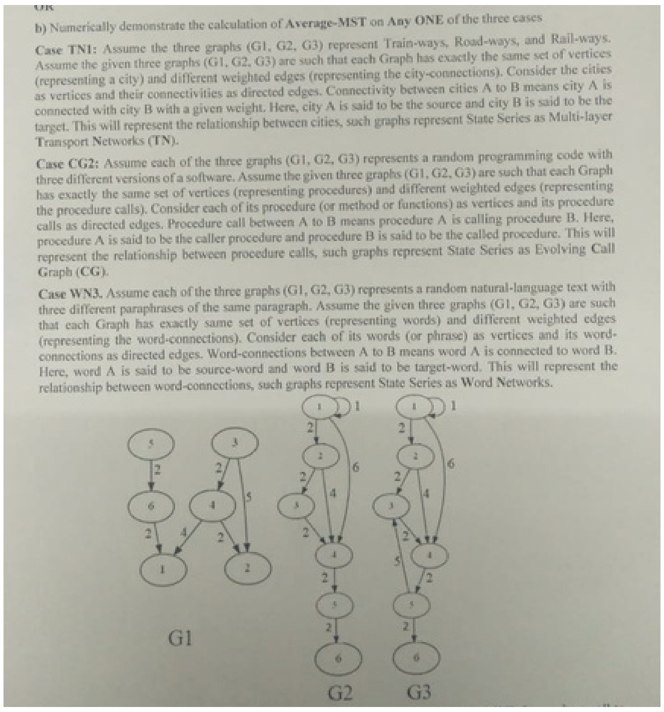
	Mean	Median	Minimum	Maximum
Prim	72897.25	15046	60	261437
Kruskal	72897.25	15046	60	261437
BellmanFord	3.75	3.5	2	6
Diijkstra	3.75	3.5	2	6

Difference In Result-:

In our project, we evaluated four different algorithms (Prim, Kruskal, Bellman Ford, and Dijkstra) on their performance metrics (Mean, Median, Minimum, and Maximum) for a specific task whereas, the results of the reference paper's findings focuses on DeceptionRank's performance in detecting deceivers from video clips and reported AUROC with confidence intervals.

```
MST:
Graph 1:60.00000
Graph 2:22469.00000
Graph 3:7623.00000
Graph 4:261437.00000
SHORTEST PATH:
Graph 1(Resistance):
0 to 6 ( 2.00)  0->7  1.00   7->6  1.00
Graph 2(Faceook):
0 to 21808 ( 2.00)  0->18427  1.00   18427->21808  1.00
Graph 3(Lastfm social):
0 to 7621 ( 6.00)  0->747  1.00   747->2020  1.00   2020->5564  1.00   5564->6108  1.00   6108-
>6146  1.00   6146->7621  1.00
Graph 4(Wikipedia-Chameleon):
2034 to 1356 (2034.00)  2034->1356 2034.00 or 2034 to 1 ( 5.00)  2034->1939  1.00   1939->1
 4.00
```

Mid-Sem Question 5 -:



OR

b) Numerically demonstrate the calculation of **Average-MST** on **Any ONE** of the three cases

Case TN1: Assume the three graphs (G1, G2, G3) represent Train-ways, Road-ways, and Rail-ways. Assume the given three graphs (G1, G2, G3) are such that each Graph has exactly the same set of vertices (representing a city) and different weighted edges (representing the city-connections). Consider the cities as vertices and their connectivities as directed edges. Connectivity between cities A to B means city A is connected with city B with a given weight. Here, city A is said to be the source and city B is said to be the target. This will represent the relationship between cities, such graphs represent State Series as Multi-layer Transport Networks (TN).

Case CG2: Assume each of the three graphs (G1, G2, G3) represents a random programming code with three different versions of a software. Assume the given three graphs (G1, G2, G3) are such that each Graph has exactly the same set of vertices (representing procedures) and different weighted edges (representing the procedure calls). Consider each of its procedure (or method or functions) as vertices and its procedure calls as directed edges. Procedure call between A to B means procedure A is calling procedure B. Here, procedure A is said to be the caller procedure and procedure B is said to be the called procedure. This will represent the relationship between procedure calls, such graphs represent State Series as Evolving Call Graph (CG).

Case WN3. Assume each of the three graphs (G1, G2, G3) represents a random natural-language text with three different paraphrases of the same paragraph. Assume the given three graphs (G1, G2, G3) are such that each Graph has exactly same set of vertices (representing words) and different weighted edges (representing the word-connections). Consider each of its words (or phrase) as vertices and its word-connections as directed edges. Word-connections between A to B means word A is connected to word B. Here, word A is said to be source-word and word B is said to be target-word. This will represent the relationship between word-connections, such graphs represent State Series as Word Networks.

MST

Graph1:
4-2 2.00000
3-4 2.00000
4-1 4.00000
5-6 2.00000
6-1 2.00000
12.00000

Graph2:
1-2 2.00000
2-3 2.00000
3-4 2.00000
4-5 2.00000
3-6 2.00000
10.00000

Graph3:
1-2 2.00000
2-3 2.00000
4-3 2.00000
4-5 2.00000
5-6 2.00000
10.00000

	Mean	Median	Minimum	Maximum
Prim	10.67	10	10	12
Kruskal	10.67	10	10	12
BellmanFord	7.34	6	6	10
Diijkstra	7.34	6	6	10

Shortest Path

```
Graph1:
3 to 0          no path
3 to 1 ( 6.00)  3->4  2.00   4->1  4.00
3 to 2 ( 4.00)  3->4  2.00   4->2  2.00
3 to 3 ( 0.00)
3 to 4 ( 2.00)  3->4  2.00
3 to 5          no path
3 to 6          no path
```

```
Graph2:
1 to 0 no path
1 to 1 ( 0.00)
1 to 2 ( 2.00) 1->2 2.00
1 to 3 ( 4.00) 1->2 2.00 2->3 2.00
1 to 4 ( 6.00) 1->4 6.00
1 to 5 ( 8.00) 1->4 6.00 4->5 2.00
1 to 6 ( 6.00) 1->2 2.00 2->3 2.00 3-
>6 2.00
```

```
Graph3:
1 to 0 no path
1 to 1 ( 0.00)
1 to 2 ( 2.00) 1->2 2.00
1 to 3 ( 4.00) 1->2 2.00 2->3 2.00
1 to 4 ( 6.00) 1->4 6.00
1 to 5 ( 8.00) 1->4 6.00 4->5 2.00
1 to 6 (10.00) 1->4 6.00 4->5 2.00 5->6 2.00
```

"Java Implementation of Student Linked List Operations"

```
public class Student_Structure {

   static class Student {
      int id;
      String name;
      Student next;

      public Student(int id, String name) {
         this.id = id;
         this.name = name;
         this.next = null;
      }
   }

   static class StudentLinkedList {
      private Student head;

      public StudentLinkedList() {
         this.head = null;
      }

      public void insert(Student student) {
         if (head == null) {
            head = student;
         } else {
            Student temp = head;
            while (temp.next != null) {
               temp = temp.next;
            }
            temp.next = student;
         }
      }

      public void traverse() {
         Student temp = head;
         while (temp != null) {
            System.out.println("ID: " + temp.id + ", Name: " + temp.name);
            temp = temp.next;
         }
      }
```

```
public void delete(int id) {
        if (head == null) {
            return;
        }
        if (head.id == id) {
            head = head.next;
            return;
        }
        Student temp = head;
        while (temp.next != null && temp.next.id != id) {
            temp = temp.next;
        }
        if (temp.next != null) {
            temp.next = temp.next.next;
        }
    }

    public Student concatenate(Student head2) {
        if (head == null) {
            return head2;
        }
        Student temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = head2;
        return head;
    }
  public void insertSorted(Student element) {
        if (head == null || element.id < head.id) {
            element.next = head;
            head = element;
            return;
        }
        Student temp = head;
        while (temp.next != null && element.id > temp.next.id) {
            temp = temp.next;
        }
        element.next = temp.next;
        temp.next = element;
    }

    public void insertQueue(Student element) {
        if (head == null) {
            head = element;
        } else {
```

```
Student temp = head;
            while (temp.next != null) {
               temp = temp.next;
            }
            temp.next = element;
         }
      }

      public Student deleteQueue() {
         if (head == null) {
            return null;
         }
         Student deletedNode = head;
         head = head.next;
         deletedNode.next = null;
         return deletedNode;
      }
   }

   public static void main(String[] args) {
      StudentLinkedList list = new StudentLinkedList();

      // Test inserting students
      list.insert(new Student(1, "John"));
      list.insert(new Student(2, "Alice"));
      list.insert(new Student(3, "Bob"));

      // Test traversing the list
      System.out.println("Students in the list:");
      list.traverse();

      // Test deleting a student
      list.delete(2);
      System.out.println("After deleting student with ID 2:");
      list.traverse();

      // Test concatenating two lists
      StudentLinkedList list2 = new StudentLinkedList();
      list2.insert(new Student(4, "Eve"));
      list2.insert(new Student(5, "Mike"));

      System.out.println("Students in the second list:");
      list2.traverse();
```

```
        list.concatenate(list2.head);
        System.out.println("Concatenated list:");
        list.traverse();

        // Test inserting an element in sorted order
        Student newStudent = new Student(6, "Sarah");
        list.insertSorted(newStudent);
        System.out.println("After inserting new student in sorted order:");
        list.traverse();

        // Test insertQueue and deleteQueue
        StudentLinkedList queueList = new StudentLinkedList();
        queueList.insertQueue(new Student(10, "Amy"));
        queueList.insertQueue(new Student(11, "Mark"));

        System.out.println("Students in the queue list:");
        queueList.traverse();

        Student deletedNode = queueList.deleteQueue();
        System.out.println("Deleted node from the queue: ID: " + deletedNode.id + ", Name: " +
deletedNode.name);
        System.out.println("Updated queue list:");
        queueList.traverse();
    }
}
```

Output-:

```
Students in the list:
ID: 1, Name: John
ID: 2, Name: Alice
ID: 3, Name: Bob
After deleting student with ID 2:
ID: 1, Name: John
ID: 3, Name: Bob
Students in the second list:
ID: 4, Name: Eve
ID: 5, Name: Mike
Concatenated list:
ID: 1, Name: John
ID: 3, Name: Bob
ID: 4, Name: Eve
ID: 5, Name: Mike
After inserting new student in sorted order:
ID: 1, Name: John
ID: 3, Name: Bob
ID: 4, Name: Eve
ID: 5, Name: Mike
ID: 6, Name: Sarah
Students in the queue list:
ID: 10, Name: Amy
ID: 11, Name: Mark
Deleted node from the queue: ID: 10, Name: Amy
Updated queue list:
ID: 11, Name: Mark
```