Name: Tan Hao Yang SID: 12212027

1. Introduction

This project addresses the Information Exposure Maximization (IEM) problem, a crucial challenge in social influence analysis. The IEM problem stems from real-world issues such as the formation of echo chambers and filter bubbles in social networks, where individuals are exposed only to like-minded information, thereby isolating them from differing viewpoints. The aim of IEM is to minimize this effect by selecting two sets of users (campaigns) from a social network to maximize the exposure to diverse information. The problem is theoretically complex, classified as NP-hard, making it computationally difficult to solve optimally.

In practical terms, the IEM problem has wide applications in social media marketing, political campaign strategy, and public health messaging, where it is crucial to spread information efficiently across various segments of a population. The social network is modelled as a graph where users are represented by nodes, and social links between them by edges. The goal is to find two balanced seed sets for two campaigns such that the overall information exposure is maximized, either by reaching users with both campaigns or by leaving them unaffected by both.

The purpose of this project is to implement and compare different algorithmic approaches to solve the IEM problem. Specifically, the project focuses on three main tasks: (1) Monte Carlo estimation to estimate the objective value, (2) the use of the Greedy Best-First Search algorithm to find the optimum balanced seed set, and (3) an Evolutionary Algorithm to explore alternative solutions for the problem. These methods aim to address the computational complexity of IEM and find high-quality solutions within acceptable time limits.

2. Preliminary

The Information Exposure Maximization (IEM) problem can be formally described using a social network, which is represented as a directed graph $G=(V,E)$, where V is the set of nodes (users) and E is the set of edges (social links) between users. Each edge $(u,v) \in E$ is associated with a diffusion probability for each of two campaigns, $p1(u,v)$ and $p2(u,v)$,

representing the likelihood that information from campaign 1 and campaign 2, respectively, propagates along that edge.

Problem Formulation:

The IEM problem involves two campaigns, c1 and c2, which seek to maximize the information exposure in a social network. Each campaign starts with an Initial Seed Set $I_i \subseteq V$ for ci, where $i \in \{1,2\}$, and the objective is to select Balanced Seed Sets $S1 \subseteq V$ for c1 and $S2 \subseteq V$ for c2, such that the total size of the two seed sets does not exceed a given budget k, i.e., $|S1|+|S2| \leq k$.

The goal is to maximize the number of users either reached by both campaigns or unaffected by both. This can be expressed mathematically as:

$$\max \Phi(S_1, S_2) = \max \mathbb{E}\left[\left|V \setminus \left(r_1(I_1 \cup S_1) \triangle r_2(I_2 \cup S_2)\right)\right|\right]$$

$$\text{s.t.} \quad |S_1| + |S_2| \leq k$$

$$S_1, S_2 \subseteq V,$$

where:

- ri(U) represents the set of users reached by campaign cic_ici using the seed set U=Ii ∪ Si, based on a diffusion process governed by the Independent Cascade Model (IC model).
- △ denotes the symmetric difference between the two sets of reached users.
- E[·] represents the expected value over multiple stochastic runs of the diffusion process.

Terminology and Notation:

- Social Network (Graph): G=(V,E), where V is the set of users and E the set of connections between users.
- Campaigns: C={c1,c2}, each representing a viewpoint or a set of information to be spread.
- Initial Seed Set: $I1, I2 \subseteq V$, the initial sets of users selected by campaigns c1 and c2, respectively.

- Balanced Seed Set: $S1, S2 \subseteq V$, the target sets of users for each campaign. The objective is to find S1 and S2 such that $|S1| + |S2| \leq k$.
- Diffusion Probability: pi(u,v), the probability that user u influences user v under campaign ci.
- Diffusion Process (IC model): A probabilistic process by which users propagate information to their neighbours with a certain probability.
- Exposed Nodes: Users who are reached by a campaign during the diffusion process. These include users who were either successfully activated or attempted (but failed) to be activated.
- Symmetric Difference: $A \triangle B$, the set of elements that are in either A or B, but not in both.

3. Methodology

3.1 General Workflow

The proposed method to solve the Information Exposure Maximization (IEM) problem is divided into three key steps, each employing a different algorithm to address the problem's computational challenges:

- **Step 1: Monte Carlo Estimation**

  This step involves using a Monte Carlo estimator to approximate the objective value of the IEM problem. The Monte Carlo simulation helps estimate the expected number of vertices reached by both campaigns or left unaffected by both, based on a set of randomly generated diffusion instances.

- **Step 2: Greedy Best-First Search**

  In this step, the Greedy Best-First Search algorithm is used to find an optimal or near-optimal balanced seed set for the two campaigns. This heuristic approach aims to maximize the information exposure by iteratively selecting nodes that contribute the most to the objective function.

- **Step 3: Evolutionary Algorithm**

The third step utilizes an Evolutionary Algorithm to explore a broader solution space and further optimize the selection of seed sets. This approach mimics the process of natural selection, using operations such as selection, crossover, and mutation to evolve potential solutions over multiple generations.

3.2 Detailed Algorithm/Model Design

a. Monte Carlo Estimation

The Monte Carlo simulation works by randomly simulating the diffusion process for each campaign and calculating the number of vertices exposed to both or neither campaign. By averaging the results over multiple runs, the algorithm estimates the objective value.

Pseudo-code:

1. Input:
    - A directed graph G=(V,E) where each edge has two associated weights, p1 and p2, representing the diffusion probabilities for two campaigns.
    - Two seed sets S1 and S2 for the initial campaigns.
    - A number of simulations run, TIMES.
2. Initialization:
    - Set total objective value totalPhi=0.
3. Monte Carlo Simulation (repeated for each simulation):
    - For each simulation:
        i. Diffusion process for Campaign 1:
            1. Start with the seed set S1.
            2. Propagate the campaign in the graph using the associated edge weights p1.
            3. Track the set of exposed nodes exposed1 (nodes either activated or attempted).
        ii. Diffusion process for Campaign 2:
            1. Start with the seed set S2.
            2. Propagate the campaign in the graph using the associated edge weights p2.
            3. Track the set of exposed nodes exposed2.
        iii. Calculate Symmetric Difference:

1. Compute the symmetric difference between the two exposed sets symmetricDiff=exposed1 △ exposed2.

iv. Update Objective Value:

1. Calculate the number of nodes exposed to both or neither campaign as bothORnone=|V|−|symmetricDiff|.
2. Add bothORnone to totalPhi.

4. Final Objective Value:
   - After TIMES simulations, compute the average objective value: finalPhi=totalPhi/TIMES.
5. Output: Return the final objective value finalPhi.

b. Greedy Best-First Search Algorithm

The Greedy Best-First Search algorithm selects nodes incrementally, aiming to maximize the marginal gain in the objective function at each step. It balances the seed sets for both campaigns while adhering to the budget constraint.

Pseudo-code:

1. Input:
   - A directed graph G=(V,E), where each edge has two associated weights p1 and p2, representing the diffusion probabilities for two campaigns.
   - Two initial seed sets S1 and S2 for campaigns 1 and 2.
   - A total budget k specifying the maximum combined size of the final seed sets.
   - Number of Monte Carlo simulations, TIMES.
2. Initialization:
   - Initialize empty seed sets S1 and S2 for both campaigns.
   - Initialize two arrays h1Values and h2Values of size |V| (number of nodes) to store the estimated gain in objective value for each node under campaigns 1 and 2.
   - Initialize counters for each campaign.
3. Main Loop (continue until the total size of S1 and S2 reaches the budget k):
   1. For each node in the graph, perform the following steps for each Monte Carlo simulation TIMES:

- Simulate the diffusion process for campaign 1 from S1 and campaign 2 from S2.
- Calculate the number of nodes exposed by both or neither campaign (denoted as bothORnone).

2. Evaluate campaign 1:
   - For each node v not in the currently activated set for campaign 1:
     - Simulate the diffusion process for campaign 1 starting from v and merge it with the existing exposed set S1.
     - Compute the symmetric difference between the new and old sets.
     - Update h1Values[v] by calculating the gain in the objective function bothORnone for campaign 1.
3. Evaluate campaign 2:
   - For each node v not in the currently activated set for campaign 2:
     - Simulate the diffusion process for campaign 2 starting from v and merge it with the existing exposed set S2.
     - Compute the symmetric difference between the new and old sets.
     - Update h2Values[v] by calculating the gain in the objective function bothORnone for campaign 2.
4. Node Selection:
   - After TIMES iterations, average the values of h1Values and h2Values.
   - Select the node with the maximum value from h1Values or h2Values.
   - If the maximum value is from h1Values, add the corresponding node to S1 and update the initial seed set for campaign 1.
   - If the maximum value is from h2Values, add the corresponding node to S2 and update the initial seed set for campaign 2.
   - Mark the selected node as used by setting its value in h1Values or h2Values to $-\infty$ (so it won't be selected again).

4. Output: Return the final seed sets S1 and S2 for both campaigns.

c. Evolutionary Algorithm

The Evolutionary Algorithm is designed to explore a large search space by evolving a population of candidate solutions. Each candidate solution represents a pair of seed sets. The algorithm iteratively improves the solutions by applying selection, crossover, and mutation.

Pseudo-code:

1. Input:
    - A directed graph G=(V,E) where each edge has two associated weights p1 and p2, representing the diffusion probabilities for two campaigns.
    - Two initial seed sets S1 and S2 for the campaigns.
    - A total budget k specifying the maximum combined size of the final seed sets.
2. Initialization:
    1. Define parameters:
        - Number of simulations TIMES=5
        - Population size POPULATION_SIZE=30
        - Number of generations NUM_GENERATIONS=10
        - Tournament size for selection TOURNAMENT_SIZE=3
        - Crossover rate CROSSOVER_RATE=0.8
        - Mutation rate MUTATION_RATE=0.01
    2. Create an initial population of seed sets, where each individual represents a potential solution (two seed sets S1 and S2).
        - Each individual is a bit vector of size 2V where V is the number of nodes in the graph.
        - The first V bits represent seed set 1, and the next V bits represent seed set 2.
        - Randomly initialize the population ensuring the total size of the selected nodes for both campaigns does not exceed the budget.

3. Evaluate Fitness:

    1. For each individual in the population:
        - Extract seed sets S1 and S2 based on the bit vector.
        - If the total size of S1 and S2 exceeds the budget, assign a negative fitness value.

        - Otherwise, evaluate the fitness using the evaluateFitness function, which:
            - Simulates the diffusion process for both campaigns using the Independent Cascade model.
            - Computes the objective function ϕ\phiϕ as the number of nodes exposed to both or neither campaign.
    2. Store the fitness values for all individuals.

4. Main Evolution Loop (Repeat for NUM_GENERATIONS):

    1. Selection:
        - Use tournament selection to select parents for crossover.
    2. Crossover:
        - With probability CROSSOVER_RATE, perform a two-point crossover on selected parents to generate two offspring.
    3. Mutation:
        - For each offspring, perform bit-flip mutation on the bit vector with probability MUTATION_RATE.
    4. Repair:
        - Ensure that the offspring's seed sets satisfy the budget constraint by randomly removing nodes if the combined size of S1 and S2exceeds the budget.
    5. Fitness Evaluation:
        - Evaluate the fitness of the new offspring using the evaluateFitness function.
    6. Elitism:
        - Combine the parent population and the new offspring.
        - Sort individuals by fitness and select the top POPULATION_SIZE individuals for the next generation.
    7. Progress:
        - Track the best fitness value at each generation.

5. Return Solution:

- After the final generation, return the individual with the highest fitness as the best solution.
- Extract and return the corresponding seed sets S1 and S2.

6. Output:

- Write the best seed sets S1 and S2 to the output file.

3.3 Analysis

a. Optimality

- Monte Carlo Estimation provides an approximation of the objective value but is not guaranteed to find the optimal solution. Its accuracy improves with the number of simulations but requires a balance between computational cost and estimation precision.
- Greedy Best-First Search is a heuristic algorithm, meaning it does not guarantee an optimal solution. However, it is computationally efficient and can quickly find near-optimal solutions, especially in large social networks.
- Evolutionary Algorithm offers a more flexible and explorative approach to finding solutions. It can potentially find better solutions than the greedy algorithm but requires more computational resources due to the nature of evolutionary operations.

b. Complexity

- Monte Carlo Estimation has a time complexity of $O(n \cdot t)$, where n is the number of nodes and t is the number of simulations.
- Greedy Best-First Search has a time complexity of $O(k \cdot n \cdot m)$, where k is the budget, n is the number of nodes, and m is the number of edges.
- Evolutionary Algorithm has a time complexity of $O(g \cdot p \cdot n)$, where g is the number of generations, p is the population size, and n is the number of nodes.

c. Performance Factors

The performance of these algorithms depends on several factors:

- Monte Carlo Estimation: The number of simulations is the key factor influencing its accuracy and computational time.

- Greedy Best-First Search: The selection of nodes and the size of the seed sets significantly impact its effectiveness.
- Evolutionary Algorithm: The population size, the number of generations, and the mutation/crossover rates are the main factors determining its performance. The algorithm's ability to escape local optima is also critical.

4. Experiments

4.1. Setup

In this project, we used synthetic datasets representing social networks for evaluating the performance of the algorithms. The datasets consist of directed graphs with varying numbers of vertices (nodes) and edges (connections between users). The edges are associated with two distinct probabilities, representing the likelihood of information diffusion for two separate campaigns. These datasets were generated to mimic real-world social networks, where information propagation occurs in a heterogeneous manner.

Software/Hardware Configuration:

- Programming Language: Python 3.10
- Libraries Used:
    - NumPy 1.24.4
    - NetworkX 2.8.8
    - Pymoo 0.6.0.1 (for evolutionary algorithms)
- Hardware:
    - CPU: 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz
    - RAM: 16 GB DDR4
    - OS: Windows 11

The experiments were run in this environment, and the performance of the algorithms was measured based on their execution time and the quality of the solutions they produced (in terms of maximizing the information exposure).

4.2 Results

a. Performance Measures:

We used two primary performance metrics:

- Running Time: The time taken by each algorithm to compute a solution.
- Optimality (Objective Value): The final value of the objective function, i.e., the number of vertices exposed to both or neither campaign.

b. Effect of Different Components and Hyperparameters (Case 0):

Experiment	Algorithm	Population Size	Mutation Rate	Time (sec)	Objective Value
1	Greedy Algorithm	N/A	N/A	6.52	442.34
2	Evolutionary Alg.	20	0.01	9.83	446.4
3	Evolutionary Alg.	30	0.01	14.25	447.8
4	Evolutionary Alg.	30	0.05	13.85	448.4
5	Evolutionary Alg.	50	0.01	23.84	450.6

Analysis of Results:

- The **Greedy Best-First Search** algorithm performed well in terms of time efficiency but slightly lagged in terms of optimality when compared to the evolutionary algorithm. This is expected, as greedy algorithms tend to converge to local optima.
- The **Evolutionary Algorithm** produced better results as the population size increased, with a noticeable improvement in the objective value. However, this came at the cost of longer running times, especially when the population size was larger.

- The **mutation rate** also impacted the performance of the evolutionary algorithm. A higher mutation rate (0.05) improved the objective value slightly but did not significantly affect the overall result.
- **Running time** was generally higher for the evolutionary algorithm, especially when hyperparameters such as population size were increased. This is expected due to the iterative nature of evolutionary algorithms and their need to evaluate a large population over multiple generations.

4.3 Analysis

The experimental results were generally in line with our expectations. The **Greedy Best-First Search** algorithm was faster, as predicted, but less capable of exploring the solution space fully, which led to suboptimal solutions in some cases. The **Evolutionary Algorithm**, on the other hand, was able to produce higher-quality solutions by balancing exploration and exploitation of the search space through selection, crossover, and mutation.

a. Effect of Components and Hyperparameters:

- **Population Size**: As expected, increasing the population size led to better solutions, as it allowed for more diverse exploration of the solution space. However, this also increased the computational time significantly.
- **Mutation Rate**: A higher mutation rate slightly improved the results by introducing more diversity into the population, preventing premature convergence to suboptimal solutions.

b. Comparison of Theoretical and Actual Performance:

The theoretical time complexity of the **Greedy Algorithm** was expected to be lower than that of the **Evolutionary Algorithm**, which involves evaluating fitness across a large population over several generations. This was confirmed by the running times observed in the experiments. The running time of the **Evolutionary Algorithm** grew linearly with population size, while the greedy algorithm consistently maintained a lower execution time.

Overall, the results demonstrate the trade-off between running time and optimality in the algorithms. While the greedy approach is faster, the evolutionary algorithm is more flexible and can yield better solutions when computational resources are available.

5. Conclusion

In this project, we explored the Information Exposure Maximization (IEM) problem and implemented three algorithmic approaches: Monte Carlo estimation, Greedy Best-First Search, and an Evolutionary Algorithm. Each of these methods was evaluated in terms of efficiency and solution quality, providing us with valuable insights into their respective advantages and limitations.

5.1 Advantages and Disadvantages of the Algorithms

- Monte Carlo Estimation: This method was efficient for estimating the objective value in a probabilistic manner. It provided quick approximations that were useful for evaluating seed sets. However, its reliance on repeated simulations made it less optimal for providing exact solutions and limited its scalability for larger networks.
- Greedy Best-First Search: The main advantage of the greedy approach was its computational efficiency. It produced solutions relatively quickly by incrementally selecting the best nodes. However, its primary disadvantage was its susceptibility to converging on local optima, as it failed to explore the broader solution space. While it worked well for smaller datasets, it underperformed in terms of optimality for larger and more complex graphs.
- Evolutionary Algorithm: The evolutionary algorithm proved to be more effective at finding high-quality solutions, especially in larger and more complex networks. Its flexibility in exploring a larger solution space through crossover and mutation allowed it to avoid local optima. The main drawback was its computational cost, as it required significantly more time and resources due to the need for simulating multiple generations and evaluating a large population.

5.2 Experimental Results and Expectations

The experimental results aligned well with our expectations. As anticipated, the greedy algorithm was faster but produced suboptimal results when compared to the evolutionary algorithm. The evolutionary approach, although more computationally expensive, managed to deliver better solutions, especially when hyperparameters such as population size were adjusted appropriately.

5.3 Lessons Learned

Through this project, we gained practical experience in implementing and fine-tuning algorithms for a complex problem like IEM. Key lessons learned include:

- Balancing performance and accuracy: The choice between a fast but potentially suboptimal greedy approach and a slower, more thorough evolutionary algorithm reflects the broader trade-off between speed and accuracy in algorithm design.
- Efficient Python Implementation: The use of libraries like NumPy and NetworkX significantly boosted the performance of our implementations. We learned that vectorized operations and efficient graph traversal methods are essential for handling large datasets in Python. Additionally, random sampling techniques and bit manipulation (as used in the genetic algorithm) can provide efficient ways to generate and explore solution spaces.

In conclusion, this project provided an in-depth exploration of different algorithmic strategies for solving the IEM problem. Each method has its strengths and weaknesses, and the choice of algorithm largely depends on the specific requirements of the problem (e.g., speed versus optimality).