CS303 Artifitial Intelligence 2024F Project1 Report

Ben Chen
chenb2022@mail.sustech.edu.cn

October 19, 2024

1 Introduction

1.1 Problem Description

In the era of information explosion, though the population of the world receives mountains of information from media every day, the Echo Chamber is formed thereby where a bunch of people only receive information from their groups. In order to break the Echo Chamber, we need to maximize the expected number of people who either receives from every groups or remains isolated. So in this project, we solve the Information Exposure Maximization problem (IEMP) by modeling the problem with a graph with two groups of nodes to simulate the information propagation from both campaigns.

The objective of this project is that, given a DAG with probabilities of passing information from one node to its neighbors as weights on edges, and two groups of nodes as the initial campaigns, we need to find two subsets of nodes for each campaign to maximize the expected number of nodes that either receives from both campaigns or remains oblivious.

1.2 Purpose

To solve the IEM problem, we divide the project into three steps:

1. Model the problem with a formal definition and a graph representation in Python.
2. Evaluate the size of the maximum number of nodes that either receives from both campaigns or remains oblivious from a given solution. It's not a trivial problem since a exact solution cannot be found in polynomial time.
3. Design a heuristic algorithm and an evolutionary algorithm to optimize the solution. Both algorithms should be efficient and effective.

In this report, we will formalize the problem in Section 2, introduce the pseudocode of our algorithm in Section 3, present the experiment design in Section 4, and analyze the results in Section Section 5. Since the source code might be hard to read, we will describe the algorithm in words in this report.

2 Preliminary

This section formally defines the problem with notations, models and formulates the result we want to achieve. Later in the report, we will use the same notations to describe the algorithm and results.

2.1 Notations

According to the problem description, we define the following terminologies:

Social Networks A DAG $G = (V, E)$ where V is the set of nodes and E is the set of edges. Each edge $(u, v) \in E$ has a weight $p_t(u, v) \in [0, 1]$ representing the probability of information propagated from node u to node v. Each edge has a tag $t = \{1, 2\}$ representing information from campaign 1 or 2.

Campaigns Two identical groups of nodes $C_1, C_2 \subseteq V$ representing the campaigns that would spread their opinion. Each campaign contains two sets $C_i = I_i \cup S_i$ where $i = 1, 2$.

Initial Seed Set Two subsets of nodes $I_1, I_2 \subseteq V$ representing the initial seed set of campaigns.

Balanced Seed Set Two subsets of nodes $S_1, S_2 \subseteq V$ representing the seed set of campaigns that we need to find to maximize the expected number of nodes that either receives from both campaigns or remains oblivious.

Budget The number of nodes that we can select. $|S_1| + |S_2| \leq k$.

Influence Result A subset of nodes $r(U) \subseteq V$ representing the nodes influenced by the seed set $U \subseteq V$. However, we need to find the mathematical expectation rather than the size of nodes.

2.2 Diffusion Model

We apply the Independent Cascade Model to simulate the information propagation in the social network. The model is defined as follows. Each nodes $v \in V$ has a state *active* or *inactive* representing whether the node has received the information.

Active Nodes receive information from their neighbors with and can activate their neighbors. Active nodes won't be changed to inactive.

Inactive Nodes have been attempted to be activated by their neighbors but failed. We don't consider the nodes that never be attempted to be reached, i.e., the nodes that have no active neighbors.

Result The active and inactive nodes are finally determined if both remains unchanged in a round, and finally decide the $r(U)$ in the equation below.

The major difference between active and inactive nodes is whether they will help spread the opinion to their neighbors. It's apparent that the model mimics the real-world information since the problem cares only about whether a person is reached.

2.3 Result

Given a social network $G = (V, E)$, two initial seed sets $I_1, I_2 \subseteq V$, and a budget k, we need to find two balanced seed sets $S_1, S_2 \subseteq V$ with $|S_1| + |S_2| \leq k$ to maximize the expected number of nodes that either receives from both campaigns or remains oblivious

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

$$s.t. \quad |S_1| + |S_2| \leq k, \ S_1, S_2 \subseteq V$$

The expectation is calculated by the sum of probabilities times size of nodes calculated above. Since we need to decide the activation if the probability is neither 0 nor 1, the choices are $2^{|V|}$.

3 Methodology

3.1 Evaluation Algorithm

Since computing the balanced information exposure for a given solution is NP-hard, we cannot directly iterate all the possible nodes to caculate the expected number of nodes. Instead, we use a Monte Carlo simulation as in Algorithm 1 to estimate the expected number of nodes. The key idea is to randomly emulate the information propagation to obtain samples, i.e., get a random value and test if it outweighs the edge's probability since higher probability will more likely to be activated. Idealy, when the sample size is large enough, the accumalted results divided by sample size, i.e. the average result will approach the expectation, since

$$\widehat{\Phi}\left(S_1, S_2\right) = \frac{\sum_{i=1}^{N} \Phi_{g_i}\left(S_1, S_2\right)}{N}$$

Algorithm 1: Monte-Carlo Evaluation

```
Data: Social network $G = (V, E)$, initial seed sets $I_1, I_2$, balanced seed sets $S_1, S_2$, simulation
      times $N$
Result: Expected number of nodes that either receives from both campaigns or remains
        oblivious $\Phi\mathbb{E}\left[|V \backslash (r_1(I_1 \cup S_1) \Delta r_2(I_2 \cup S_2))|\right]$
1  $E \leftarrow 0$;
2  for $i \leftarrow 0$ to $N$ do
3      Infected, Active, Activated $\leftarrow I_1 \cup S_1$;
4      while there are active nodes do
5          for $v \in$ Active do
6              for edge $(v, u) \in E$ do
7                  Infected $\leftarrow$ Infected $\cup \{v\}$;
8                  if $u \notin$ Infected and $p_1(v, u) >$ rand() then
9                      Activated $\leftarrow$ Activated $\cup \{u\}$;
10                 end
11             end
12         end
13         update Active nodes with nodes been added to Activated in this iteration;
14     end
15     Do the same for $I_2 \cup S_2$;
16     $E \leftarrow E + |V \backslash (r_1(I_1 \cup S_1) \Delta r_2(I_2 \cup S_2))|$;
17 end
18 return $E/N$;
```

3.2 Heuristic Algorithm

As taught in lecture and lab sessions, we can use a greedy best-first search algorithm. The main idea is to select the nodes that can maximize the increment of the balanced information exposure, i.e., $h(v)$. After we pick some candidate nodes, we can use greedy search to determine effective nodes to add. Before performing search, we need to apply a Monte Carlo simulation to get the candidate node sets $r_1(\{v_i\})$ and $r_2(\{v_i\})$ to reduce the search space and with some pruning. We do the following steps:

1. On input social networks $G = (V, E)$, for each campaign C_i $(i = 1, 2)$, we delete all edges with $p_i(u, v) \leq t$ for some threshold t to get subgraph G_i. We estimate the threshold to about 10^{-7} since it's much unlikely to be activated at this probability.

2. In the subgraph G_i, we perform a similar Monte Carlo simulation as in Algorithm 1 to get the candidate nodes $r_i(\{v_i\})$. But this time, we only need to simulate for a much smaller amount. Take their intersection to get the candidate nodes.

3. We then run the heuristic search algorithm as in Algorithm 2 to get the balanced seed sets, this works since we have the estimation:

$$r_1(\widehat{S_1 \cup \{v_i\}}) = r_1(S_1) \cup r_i(\{v_i\})$$

   by which we can evaluate the heuristic search. Same for r_2.

For the first and second steps, the time complexity is $O(|V| + |E|)$, since we just need to widely visit the nodes and edges. The third step is $O(k|V|)$, since we need to iterate k times to get the balanced seed sets. The overall time complexity is $O(k|V|)$ which is feasible theoretically.

Algorithm 2: Heuristic Search of IEM

```
   Data: Social network $G = (V, E)$, initial seed sets $I_1, I_2$, budget $k$, candidate nodes $r_1(\{v_i\})$
         and $r_2(\{v_i\})$
   Result: Balanced seed sets $S_1, S_2$
1  $S_1, S_2 \leftarrow \emptyset$;
2  $\Phi_{max} \leftarrow \Phi(S_1, S_2)$;
3  while $|S_1| + |S_2| < k$ do
4      $v_1^* \leftarrow \arg\max_v (\Phi(S_1 \cup \{v\}, S_2) - \Phi(S_1, S_2))$;
5      $v_2^* \leftarrow \arg\max_v (\Phi(S_1, S_2 \cup \{v\}) - \Phi(S_1, S_2))$;
6      if $\Phi(S_1 \cup \{v_1^*\}, S_2) = \Phi(S_1, S_2 \cup \{v_2^*\}) = \Phi(S_1, S_2)$ then
7          break;
8      end
9      else if $\Phi(S_1 \cup \{v_1^*\}, S_2) > \Phi(S_1, S_2 \cup \{v_2^*\})$ then
10         $S_1 \leftarrow S_1 \cup \{v_1^*\}$;
11     end
12     else
13         $S_2 \leftarrow S_2 \cup \{v_2^*\}$;
14     end
15 end
16 return $S_1, S_2$;
```

3.3 Evolutionary Algorithm

The preliminary process of Evolutionary Algorithm in this projects is similar to the Heuristic Algorithm. We need to prune the graph and get the candidate nodes $r_1(\{v_i\})$ and $r_2(\{v_i\})$ to reduce the search space. The difference is that we need to use a population to evolve the seed sets, instead of greedy search. The pseudocode is shown in Algorithm 3. In Evolutionary Algorithm, we need to

1. Initialize a population of size N with random seed sets in candidate nodes.
2. For each iteration, evaluate the fitness of each individual in population and sort them by fitness.
3. Keep the top half of the population and randomly pick $N/2$ pairs of parents to generate offspring by crossover and mutation.
4. Replace the population with the new elite population.
5. Returns the best individual in the population when the iteration reaches the maximum.

Algorithm 3: Evolutionary Algorithm of IEM

```
   Input: Initial population size $N$, budget $k$, probability distribution $probw$, max iterations
          $max\_iter$
   Output: Balanced seed sets $S_1, S_2$
1  Initialize:
2  Generate an initial population $P$ of size $N$;
3  foreach iteration $i : 1 \rightarrow max\_iter$ do
4      foreach individual $p$ in population $P$ do
5          Evaluate the fitness of $p$ using fitness(p);
6      end
7      Sort population $P$ by fitness, keeping the top $N$ individuals;
8      foreach pair of parents $(p_1, p_2)$ from top $N$ do
9          $son_1, son_2 \leftarrow$ crossover($p_1, p_2$);
10         $son_1, son_2 \leftarrow$ mutate($son_1, son_2$);
11         $P \leftarrow P \cup \{son_1, son_2\}$;
12     end
13 end
14 Return P[0];
```

Note that since the newly generated sons will not be constrained by the budget, we need to consider that in fitness evaluation, i.e., if the size of evaluated sample is oversize, `fitness` will return a negative value to avoid the selection of the individual. The `crossover` and `mutate` functions are generally implemented as taught in the lecture and are not much customized.

4 Experiments

4.1 Setup

In this section, we adapt a compound testing methods to evaluate the effectiveness of the algorithms, which is to test the same code locally and remotely on Online Judge, and compare the results. For local testing, we use the following environment:

Model	MacBook Air M3 13’
CPU	Apple M3 4E + 4P 2.4GHz-3.7GHz
Memory	LPDDR5-6400 8GB Unified Memory

And for remote testing, we use the following environment:

Model	Online Judge
CPU	Intel Xeon E5-2680 2.2GHz * 1
Memory	UNKNOWN

We use the same libraries as the Online Judge, but virtually imported `numpy == 1.24.4` in our code. And builtin libraries are `time`, `collections`, `argparser` and `functools`.

For the accuracy of evaluation, we MUST use the dataset provided by the Online Judge to test the algorithm and OJ also provides the reference result of first three datasets. Hopefully, the dataset provided contains different sizes of graphs with a clear gradient. After finishing the evaluator, for heuristic and evolutionary algorithms, we can apply to test the balanced information exposure from the balanced seed sets generated by the algorithms to evaluate the effectiveness of it.

4.2 Evaluator Results

In the evaluator experiment, we majarly test the time efficiency and the correctness on local environment since the Online Judge will onle feedback the total cost and test results. The results are shown in the table below.

Table 1: Dataset of `Evaluator`

Case	Nodes	Edges	Budget	Baseline	Result	Local Cost
#1	475	13289	10	430	423.147	1.26s
#2	36742	49248	15	35900	35562.963	1.52s

From this table, we can easily see that time cost grows with the size of nodes and edges at the same simulation times, but it’s not a linear relationship. Our initial guess is that the second dataset has a less dense graph or less probability of edges, which makes the propagation earilier to stop.

We also tested the simulation times of the Monte-Carlo algorithm starting from 2000 and keep decreasing until the result becomes unstable. The result shows that simluating for 500 times is good enough to get a stable result, which will cost averagely 1s on the local machine and totally 2s on the Online Judge for 2 datasets.

4.3 Heuristic Results

For dataset #0-2, we have the data and requirement locally so that it’s able to test the efficiency and accuracy of our heuristic algorithm. For dataset #3-4, we can only see the result since the detailed timing on OJ is not available. The results are shown in the table below.

Table 2: Dataset of `IEMP_Heur`

Case	Nodes	Edges	Budget	Baseline	Result	Local Cost
#0	475	13289	10	430	430.128	0.31s
#1	36742	49248	15	35900	35908.591	19.22s
#2	36742	29248	15	36000	36183.786	9.58s
#3	7115	103689	/	/	6923.453	/
#4	3454	32140	/	/	3411.335	/

From the table, we can see a clear trend that with greater networks comes the longer search, as in our time complexity analysis at Section 3.2. It's interesting to observe that with the same network in dataset #1 and #2, different initial nodes will lead to a different time cost and results. Dataset #2 has less initial seeds but leads to a faster search and a better result. And from our observation of the networks, we found that the initial seeds in dataset #2 have a better neighborhood, for which the balanced seeds have less impact on the balanced information exposure. Therefore, we can perform a better search with a good initial seed set.

Meanwhile, at the first step described at Section 3.2, we have a hyperparameter *simtimes* to generate a candidate node set. We tested the parameter with 3, 4, 5, 10, 50. It's uncommon to see that with larger simulation times, the algorithm runs slower but with a worse result. With 5, 10, 50 simulation times, the time cost is applicable but the result will randomly drop below the baseline. After 8 submissions to the Online Judge, we can only get from 4.1 to 5.6 score. We haven't resolved the issue yet, but we can infer that more candidate nodes will NOT lead to a better result. At the end of the experiment, we set the parameter to 3 to pass all testcases on the Online Judge.

4.4 Evolutionary Results

The datasets in Evolutionary Algorithm share the same network as in the Heuristic Algorithm and also, we can only test the first three datasets locally.

Table 3: Dataset of `IEMP_Evol`

Case	Nodes	Edges	Budget	Baseline	Result	Local Cost
#0	475	13289	10	415	440.43	0.69s
#1	13984	17319	14	13580	13738.047	2.62s
#2	13984	17319	14	13580	13700.597	2.80s
#3	3454	32140	/	/	3401.567	/
#4	3454	32140	/	/	3386.141	/

Clearly seen from the table, the efficiency of the algorithm depends on the network sizes, as the fitness evaluation takes $O(|V|)$ time and the candidate generation takes $O(k|V|)$ time.

For the population size, we initially set the population size to 100 which achieves a good result. We also tested the population size with 10, 50, 200, 500, 1000. We found that the results get better with 10, 50 and 100, but with 200, 500 and 1000, the results remains stable, i.e., since it converges to the optimal solution.

For the mutation, we use a hybrid mutation strategy with a equally distributed probability for each mutation operator. They mutation methods are Simple Mutation, Flip Mutation and Swap Mutation. For the crossover, we use a simple one-point crossover. The components work well enough to get a acceptable result.

5 Conclusion

In this projects, we have learned and implemented the algorithms to solve a NP problem, the Information Exposure Maximization problem, step by step. At the first stage, we design a Monte-Carlo

Simulation to evaluate the expected balanced information exposure. At the second and third stage, we perform a heuristic search and an evolutionary algorithm to optimize the solution. For each stage, we compared different hyperparameters and strategies to see the difference in time cost and result, and finally evaluated the algorithms on the Online Judge. Also in this report, we discuss the experiment results and possible answers to the issues we encountered, like the wired behavior of the heuristic algorithm and the population size of the evolutionary algorithm, and then give a optimal configuration in codes. Through the project, we have a better understanding of the AI algorithms and the NP problems.

5.1 Algorithm Evaluation

5.1.1 Monte-Carlo Evaluation

Advantages

- The implementation is simple and easy to write, test and understand.
- The algorithm is efficient and effective to estimate the expected number of nodes, without the need to iterate all the possiblities.

Disadvantages

- The algorithm is not reliable, since the result is an estimation. The results vary largely on local environment.
- The algorithm may be slow to converge to the expectation for a complex network.

5.1.2 Heuristic Algorithm

Advantages

- Simple and easy to implement. The greedy idea is easy to understand.
- Stable and reliable results for most cases since the search is deterministic.

Disadvantages

- The algorithm is time-consuming since it will search the whole networks.
- Few optimization strategies to our best knowledge, since it's essentially a graph algorithm.

5.1.3 Evolutionary Algorithm

Advantages

- The algorithm is efficient and effective to optimize the solution.
- Much easier to implement since it uses a general framework.

Disadvantages

- Needs to tune the hyperparameters and choose the right components to get a good result, and needs lots of experiments to test them.
- Too general to get improvements, only the parameters, mutation and crossover can be customized.

5.2 Improvement

5.2.1 Heuristic Algorithm

As it's a graph algorithm, we can use some graph optimization to improve the search like

- Cutting edges is not enough, we may also consider to prune the isolated nodes or nodes with too much neighbors.
- More advanced pruning strategies like dynamically adjusting the threshold of the edge probability to cut a moderate number of edges.

5.2.2 Evolutionary Algorithm

The efficiency is good enough, so all we need to do is to improve the results

- More complex mutation and crossover strategies like multi-point crossover and inversion mutation. And more combinations of them should be tested to get a better result.
- Different parents selection strategies like tournament selection, instead of elite selection.
- Use a gradient descent algorithm to optimize the hyperparameters.

5.3 Experiments Evaluation

The experiments are mostly consistent with our theoretical analysis and expectations. The Monte-Carlo algorithm returns a result close to the expectation and shows a convergence with larger simulation times. The Heuristic Algorithm is efficient and effective to get a good result, but the time cost is not acceptable for large networks. The Evolutionary Algorithm is efficient and effective to optimize the solution, and the tuning result of hyperparameters is reasonable.

5.4 Lessons Learned

The concepts of heuristic search and evolutionary algorithm are new to me and this project offers me a chance to actually implement them. Learning the algorithm by modelling and programming to solve a real problem with it is a good way to understand the AI algorithms. Also, the project gets me familiar with the Python programming language, such as the argument passing, the Numpy library. What's more, I learnt how to tune the hyperparameters and strategies in a AI algorithm by doing experiments and analyzing the results.