南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CS303 Project1 Report

姓名 秦恺通

学号 12212606

院系 计算机科学与工程系

2024 年 11 月 1 日

目录

1 引言

本项目研究的是**信息曝光最大化 (Information Exposure Maximization, IEM)** 问题，它源自于社交网络中的影响力传播和信息扩散领域。随着互联网和社交媒体的迅速发展，用户越来越容易受到算法推荐和兴趣过滤的影响，从而被限制在"回音室"或"信息茧房"中。这些现象导致用户只接触与自身观点一致的信息，而无法了解其他多样的观点。

IEM 的目标是通过设计合适的用户集来平衡两种活动（campaign）的信息曝光，减少回音壁效应和信息隔离。这一问题被建模为一个在社交网络中选择两个种子集的问题，以最大化节点的平衡曝光度。具体而言，**IEM** 旨在找到两个种子集，使得尽可能多的用户既能接触到两种观点，或者完全不知晓两种观点。该问题应用了**独立级联模型 (Independent Cascade Model, IC)** 模拟信息的传播过程，并且通过优化传播效果来解决影响力最大化中的平衡问题。

IEM 问题的研究起源于社交影响力分析领域，它不仅具有理论价值，而且对现实世界中的广告投放、信息传播策略制定等具有重要应用。在现实生活中，**IEM** 可以应用于多个领域：

- **信息传播与媒体管理**：帮助媒体平台或社交网络在推广不同观点时平衡用户的曝光度，减少极端化倾向。
- **营销与竞选**：企业在广告投放中，需要选择关键用户来推广不同品牌或产品，从而优化用户对多方观点的接受程度。
- **公共政策与社会治理**：在公共信息宣传中平衡不同立场的曝光，帮助社会成员更全面地了解公共议题。

在本报告中，我们将详细介绍 **IEM** 问题的形式化定义、预备概念、以及我们所设计的**启发式算法**和**遗传算法**的具体实现和性能分析。我们期望通过本项目的研究，为社交网络中信息传播的优化提供新的视角和工具。

2 问题定义

在本报告中，我们对 **IEM** 问题进行参数定义，本节将详细说明涉及的关键概念及数学符号。

2.1 社交网络模型与扩散过程

- **社交网络**：使用有向图 $G = (V, E)$ 表示，其中：
  - $V = \{v_1, v_2, \ldots, v_n\}$ 是用户节点的集合。

- $E \subseteq V \times V$ 是边的集合，每条边 $(u, v) \in E$ 表示用户 u 和用户 v 之间的社交联系。

- **Campaigns**：两个组织 $C = \{c_1, c_2\}$，分别表示持有不同观点的两个组织。
  - 每个组织 c_i 会从初始种子集 $I_i \subseteq V$ 开始传播信息。

- **扩散概率**：对于每条边 $(u, v) \in E$，定义两个独立的扩散概率 p_{uv}^1 和 p_{uv}^2：
  - p_{uv}^1 是组织 c_1 通过边 (u, v) 激活节点 v 的概率。
  - p_{uv}^2 是组织 c_2 通过边 (u, v) 激活节点 v 的概率。

- **扩散模型**：使用**独立级联模型（Independent Cascade Model, IC）**模拟信息扩散。
  - 种子集中的节点在 $t = 0$ 时被激活。
  - 每个活跃节点 u 在下一步中以概率 p_{uv}^i 尝试激活其邻居节点 v。
  - 每个节点只有一次尝试激活其他节点的机会。当无法再激活新节点时扩散终止。

3 主体代码

为了解决上述问题，本文使用启发式算法和遗传算法解决 IEM 问题，针对每次选择的 S_1 和 S_2，使用蒙特卡洛模拟来评估当前选择的两个集合的效果。蒙特卡洛模拟保存在 `Evaluator.py` 中，启发式算法在文件 `IEMP_Heur.py` 中，进化算法在文件 `IEMP_Evol.py` 中。

3.1 `Evaluator.py`

3.1.1 总体流程

该部分代码由以下几个步骤组成，首先从文件中读取并构建图结构和种子集，然后使用基于独立级联模型的扩散仿真来模拟两个观点的传播过程，通过蒙特卡洛方法多次模拟，计算两种观点在平衡曝光方面的期望值。

3.1.2 主体伪代码

Algorithm 1 Information Diffusion Simulation

Require: Graph $G = (V, E)$, Seed sets U_1 and U_2, Edge weights P_1 and P_2
Ensure: Reachable sets for both campaigns: $reach\_u1$ and $reach\_u2$

$reach\_u1 \leftarrow U_1$, $reach\_u2 \leftarrow U_2$
$q_1 \leftarrow$ deque(U_1), $q_2 \leftarrow$ deque(U_2)
$in\_q1 \leftarrow \emptyset$, $in\_q2 \leftarrow \emptyset$
while q_1 is not empty **do**
    $current \leftarrow q_1$.popleft()
    if $current \notin in\_q1$ **then**
        $in\_q1 \leftarrow in\_q1 \cup \{current\}$
        for each $neighbor$ of $current$ in G **do**
            if random() $< P_1(current, neighbor)$ **then**
                q_1.append($neighbor$)
            end if
            $reach\_u1 \leftarrow reach\_u1 \cup \{neighbor\}$
        end for
    end if
end while
while q_2 is not empty **do**
    Repeat the same process for U_2 using weights P_2
end while
return $reach\_u1, reach\_u2$

Algorithm 2 Monte Carlo Simulation

Require: same as Simulation(G, U_1, U_2, P_1, P_2)
Ensure: Expected balanced exposure value

$nodes \leftarrow V(G)$ ▷ Get the set of all nodes in the graph
for $i \leftarrow 1$ to N **do**
    $u_1, u_2 \leftarrow$ Simulation(G, U_1, U_2, P_1, P_2)
    $\Phi_i \leftarrow |nodes \setminus (u_1 \Delta u_2)|$
end for
return $\frac{\sum_1^N \Phi_i}{N}$

3.1.3 性能分析

使用蒙特卡洛方法提供期望值的近似解，虽然不保证全局最优，但随着迭代次数的增加，结果的质量会逐渐提高。并且由于仿真过程中存在随机性，不同运行之间的结果可能会有所不同。模拟扩散过程的时间复杂度是 $O(n+m)$，若仿真进行 N 次，总复杂度为 $O(N \cdot (n+m))$。

经过本次项目后续部分的实验，这个部分的代码效率在这些部分可以提升:

- 使用 `np.random.rand()` 代替 `random.random()`
- 将判断当前节点是否扩散放在遍历邻居里面: 使用一个集合判断是否完成扩散，如果没有完成扩散，且传播概率符合条件，才会把节点加入队列。
- 计算 Φ 的时候，由于只需要得到数量。如果使用集合操作，需要查看集合中每个元素是否在 $r_1\Delta r_2$ 中，需要遍历整个图的节点，即使集合使用哈希表实现，也会消耗很多时间。所以只需要计算 $|r_1\Delta r_2|$，$\Phi = |V| - |r_1\Delta r_2|$，这样只需遍历小集合的节点，提高了运行效率。

3.2 `IEMP_Heur.py`

3.2.1 总体流程

Greedy Best-First Algorithm 是一种基于启发式搜索的方法，旨在解决 **IEM** 问题。在社交网络中，该算法通过选择每次增加一个节点来最大化两种观点的影响效果。

3.2.2 主体伪代码

Algorithm 3 Node List Generation

Require: Graph $G = (V, E)$

Ensure: Sorted list of nodes and check number

```
n ← G.number_of_nodes()                                   ▷ Get the number of nodes
node_list ← list(G.nodes)                                 ▷ Get the list of nodes
node_list.sort(key = lambda x : G.out_degree(x), reverse = True)  ▷ Sort nodes by out-degree
if n > 10000 then
    return node_list[: 300], 10          ▷ Return top 300 nodes and check number 10
else if n > 5000 then
    return node_list, 3                  ▷ Return all nodes and check number 3
else
    return node_list[: 100], 5           ▷ Return top 100 nodes and check number 5
end if
```

Algorithm 4 Greedy Best-First Algorithm

Require: Graph $G = (V, E)$, Initial seed sets I_1 and I_2, Budget *budget*

Ensure: Selected seed sets S_1 and S_2

$S_1, S_2 \leftarrow \emptyset, \emptyset$
$n \leftarrow G.number\_of\_nodes()$
$nodes, check\_num \leftarrow \text{node\_list}(G)$
while $|S_1| + |S_2| < budget$ **do**
  $h_1 \leftarrow \text{zeros}(n)$, $h_2 \leftarrow \text{zeros}(n)$ ▷ initialization
  $U_1 \leftarrow I_1 \cup S_1$, $U_2 \leftarrow I_2 \cup S_2$
  for each j from 1 to $check\_num$ **do**
    $r_1, r_2 \leftarrow \text{simulation}(G, U_1, U_2)$
    $start \leftarrow n - |r_1 \Delta r_2|$ ▷ get Φ without adding v
    for each v in $nodes$ **do**
      $r1\_v \leftarrow \text{bfs}(G, v, 'weight1') \cup r_1$
      $end \leftarrow n - |r1\_v \Delta r_2|$ ▷ get Φ with adding v to S1
      $h_1[v] \leftarrow h_1[v] + (end - start)$ ▷ compute the increments
      $r2\_v \leftarrow \text{bfs}(G, v, 'weight2') \cup r_2$
      $end \leftarrow n - |r2\_v \Delta r_1|$ ▷ get Φ with adding v to S2
      $h_2[v] \leftarrow h_2[v] + (end - start)$ ▷ compute the increments
    end for
  end for
  $v_1 \leftarrow \text{argmax}(h_1)$, $v_2 \leftarrow \text{argmax}(h_2)$ ▷ get argmax v_1, v_2
  if $h_1[v_1] > h_2[v_2]$ **then**
    $S_1 \leftarrow S_1 \cup \{v_1\}$
  else
    $S_2 \leftarrow S_2 \cup \{v_2\}$
  end if
end while
return S_1, S_2

3.2.3 性能分析

在每次循环中，假设执行 $check\_num$ 次仿真，每次仿真调用 simulation 函数的时间复杂度为 $O(|V|+|E|)$。遍历每个节点，将每个节点单独来传播，计算增量，总体复杂度为 $O(|E|)$。所以，在每次执行 while 循环内部，消耗的时间为 $O(check\_num \cdot (|V|+|E|))$。在循环外部，选择 k 个节点，所以整个算法的时间复杂度为 $O(k \cdot check\_num \cdot (|V|+|E|))$。在 `Algorithm 3 Node List Generation` 中，参数是通过实验测试得到的最优效果，原因见实验分析部分。

3.3 `IEMP_Evol.py`

3.3.1 总体流程

该算法的主要目标是使用**遗传算法**解决信息曝光最大化问题（**IEM**）。首先，创建初始种群，每个个体表示一组种子集 S_1 和 S_2。然后，在遗传算法迭代的过程中：在每一代中，计算每个个体的适应度值；使用轮盘赌选择算法选择父代个体，对选中的父代进行交叉和变异生成下一代个体。当算法迭代次数达到设定的代数后，返回这一代最优的个体。遗传算法的主体部分的伪代码如下。

3.3.2 主体伪代码

Algorithm 5 Genetic Algorithm for IEM

Require: Graph G, Initial seed sets I_1, I_2, Population size pop_size, Number of genes num_genes, Budget k, Generations $generations$, Mutation rate $mutation\_rate$, Crossover rate $cross\_over\_rate$

Ensure: Best seed sets S_1 and S_2

```
population ← initialize_population(pop_size, num_genes, k)
for each generation from 1 to generations do
    fitness_value ← [ ]
    for each individual in population do
        fitness_value.append(fitness(G, I1, I2, individual, k))
    end for
    next_population ← [ ]
    while |next_population| < pop_size do
        parent1 ← roulette_wheel_selection(population, fitness_value)
        parent2 ← roulette_wheel_selection(population, fitness_value)
        child1, child2 ← crossover(parent1, parent2, cross_over_rate)
        child1 ← mutate(child1, mutation_rate)
        child2 ← mutate(child2, mutation_rate)
        next_population.append(child1)
        next_population.append(child2)
    end while
    population ← next_population[: pop_size]
end for
best_individual ← find_best_individual(population, G, I1, I2, k)
return best_individual
```

3.3.3 性能分析

在这部分将分析使用遗传算法解决 IEM 问题的复杂度问题，假设图由 n 个节点和 m 条边组成：

1. **初始化种群**：在算法开始的时候，新建一个种群，种群中包含 pop_size 个个体，每个个体长度为 $2n$，使用二进制编码表示当前节点是否存在于 S_1 和 S_2 中。这部分的时间复杂度是 $O(pop\_size \cdot n)$，用来存储种群的空间复杂度是 $O(pop\_size \cdot n)$。

2. **适应度计算**：由于最终需要得到 $|S_1| + |S_2| = k$，所以当个体中 1 的数量不等于 k，设置适应度为 $-(|S_1| + |S_2|)$；当个体中 1 的数量等于 k，就使用集合 S_1 S_2 来计算 Φ 作为该个体的适应度（这里使用蒙特卡洛算法提高 Φ 的准确度），这里的时间复杂度为 $O(pop\_size \cdot check\_num \cdot (n + m))$。

3. **选择、交叉和变异**：使用轮盘赌算法的时间复杂度是 $O(pop\_size)$，交叉和变异的时间复杂度是 $O(n)$。

从上述分析中可以发现，假设算法迭代次数为 $generations$，该算法总体的时间复杂度为 $O(generations \cdot pop\_size \cdot check\_num \cdot (n + m))$。

4 实验部分

4.1 实验设置

在本项目中，使用的数据集是从 `blackboard` 上下载的 `datasets`。这些数据集有着不同的规模和结构属性，为贪心优先算法和遗传算法提供了良好的测试基础。

dataset	节点数量	边数量
dataset1	475	13,289
dataset2	36,742	49,248
dataset3	13,984	17,319

实验在以下环境进行：

- **软件环境**：`python==3.10.15`，`numpy==1.24.4`，`networkx==2.8.8`
- **硬件环境**：
  - CPU: 12th Gen Intel(R) Core(TM) i5-12500H 2.50 GHz
  - RAM: 16.0 GB
  - OS: Windows 11

4.2 实验结果

经过实验得到以下结果:

表 1: `IEMP_Heur.py`

dataset	**result**	**runtime**
dataset1	453.439	4.812 s
dataset2	36008.511	171.589 s
dataset3	36045.963	206.056 s

表 2: `IEMP_Evol.py`

dataset	**result**	**runtime**
dataset1	420.361	36.573 s
dataset2	13591.848	83.677 s
dataset3	13369.576	130.785 s

- **在贪心优先算法中**，选择节点的集合大小对算法的时间和表现有着显著的影响。经过实验多种选择节点的集合大小，比如按照每个节点的出度排序、按照每个节点的 $weight_1$ 之和排序、按照每个节点 `bfs` 遍历接触到的节点个数排序等等，然后根据图节点总数选取一定比例的节点作为选取节点的集合。我通过 oj 平台测试这些排序方法，发现按照每个节点出度排序，实验效果最好。
- **在遗传算法中**，种群大小对性能有显著影响。较大的种群大小的确能够找到更好的解，但运行时间也随之增加。但调整计算适应度函数的验证次数却对实验结果产生了显著的影响。

4.3 实验分析

这些实验结果说明算法有良好的准确性和高的效率。通过 online judge 平台的测试，进一步评估并验证了算法的鲁棒性。

在**贪心优先算法**的实验中，选择节点的集合大小对算法的时间和表现有着显著的影响。经过 oj 平台测试算法，我发现通过出度排序，选取出度最多的一些节点，效果很好，可以拿到 5.3 的分数。这是因为出度较大的节点可以向更多的邻居传播信息。由于每条边的权重较小，通常在 `bfs` 的第三层及以后，传播的概率会显著降低，几乎无法传播到第六层以外。因此，选择节点时应集中在出度最多的节点上。同时，通过增加蒙特卡洛实验的次数，可以提高结果的普遍性，降低随机算法的偶然性。此外，这种方法能够**显著提升运行效率**，结果经过多次蒙特卡洛模拟验证，证明可以有效最大化 Φ。选取节点的算法伪代码详见Algorithm 3。

但是上述算法有一定的局限性，需要适应图的结构，由于不同结构的图表现不同，因此该算法的鲁棒性并不理想，也没有通过 oj 平台的鲁棒性测试。尽管出度较大的节点确实能够传播给更多的邻居，但在第一张鲁棒性测试图中，这种选择方式表现较差，结果约为 6790。因此，扩大出度节点的选择范围是一个解决方案，最终发现，只有考虑所有节点，才能有效克服这一问题。

在**遗传算法**的实验中，我们发现种群数量和迭代次数的微调对最终结果的影响相对较小。然而，调整计算适应度（**fitness**）函数的验证次数却对实验结果产生了显著的影响。这是因为这样可以降低随机算法的偶然性，从而正确的选择表现好的亲本遗传下去。具体来说，适应度函数是遗传算法中至关重要的一环，它决定了每个个体在种群中的生存和繁殖机会。当我们增加验证次数时，能够更准确地评估个体的实际表现。这种增强的评估不仅提升了选择的可靠性，还使得优秀个体得以在下一代中更好地传承其优良特性。我们应当更加重视适应度函数的计算策略，通过合理配置验证次数来提升算法的整体性能。

5 得出结论

本研究围绕**信息曝光最大化（IEM）**问题展开，通过设计和实现两种算法——贪心优先算法和遗传算法，对社交网络中的信息传播进行了深入分析。实验结果表明：

1. **算法性能**：在不同数据集上，贪心优先算法表现出较高的运行效率和准确性，尤其是在节点选择策略上，通过对节点出度的排序，成功优化了信息传播效果。而遗传算法在处理复杂性和适应度评估方面展现了其优势，尽管运行时间相对较长，但在数据集 2 和数据集 3 上也取得了令人满意的结果。

2. **节点选择策略**：实验显示，节点选择的集合大小以及选择策略对算法性能有显著影响。贪心优先算法通过选择出度较大的节点来提高信息传播的效率，这一发现为未来在实际应用中优化信息传播策略提供了重要依据。

3. **适应度函数的重要性**：在遗传算法中，适应度函数的计算次数对最终结果有显著影响。增加验证次数能够降低随机性，从而更准确地评估个体表现。这一发现强调了在遗传算法设计中，如何合理配置适应度评估策略的重要性。

4. **局限性与未来研究方向**：尽管该完成项目的过程中取得了一定成果，但仍存在局限性。例如，贪心优先算法在某些图结构中表现不佳，提示我们需要进一步探索更具鲁棒性的节点选择方法。此外，未来的研究可以考虑结合其他机器学习技术，以提高 IEM 问题解决方案的普遍适用性和效果。

综上所述，本项目不仅为 IEM 问题提供了有效的解决方案，也为社交网络中的信息传播优化提供了新的视角和工具。希望未来能在此基础上进行更深入的研究，以推动该领域的发展。

参考文献

[1] K. Garimella, A. Gionis, N. Parotsidis, and N. Tatti. Balancing information exposure in social networks. In *NeurIPS*, pages 4663–4671, 2017.

[2] S. Cheng, H. Shen, J. Huang, E. Chen, and X. Cheng. Imrank: influence maximization via finding self-consistent ranking. In *SIGIR*, pages 475–484, 2014.

[3] M. Gong, J. Yan, B. Shen, L. Ma, and Q. Cai. Influence maximization in social networks based on discrete particle swarm optimization. *Information Sciences*, 367-368:600–614, 2016.

[4] Q. Jiang, G. Song, G. Cong, Y. Yeang, E. Si, and K. Xie. Simulated annealing based influence maximization in social networks. In *AAAI*, 2011.