

电子科技大学计算机科学与工程学院

# 标准实验报告

(实验) 课程名称 数据结构与算法

# 电子科技大学

# 实验报告

学生姓名：陶浩轩 学号：2023080902011 指导教师：陈端兵

## 1. 实验室名称：

学知三组团 110139

## 二、实验项目名称：

重要节点组挖掘

## 三、实验原理：

本实验基于复杂网络理论，利用网络中节点的度以及节点之间的连接关系，寻找对信息传播影响最大的节点组。主要原理如下：

- **网络模型构建**：将用户之间的关系抽象为网络，每个用户作为一个节点，用户之间的联系作为边。
- **节点重要性评估**：
  - **度排序**：节点的度越大，其连接的节点越多，对信息传播的影响力也越大。
  - **基于投票策略**：节点根据其邻居节点的投票能力进行排序，投票能力高的节点更有可能被选中作为种子节点，从而启动信息传播。本实验中，投票能力采用 PageRank 算法计算，PageRank 算法通过分析节点之间的链接关系，评估节点在网络中的重要性。
- **信息传播模型**：采用 SIR 模型模拟信息在网络中的传播过程，评估不同节点组作为种子节点时的信息传播效果。

## 四、实验目的：

- 探索在有限资源下，如何选择一组节点进行营销推广，以达到最佳效果。

- 比较度排序策略和基于投票策略在节点组选择上的优劣。
- 分析不同初始种子节点数量和感染率对信息传播效果的影响。

## 五、实验内容：

1. 网络构建：利用邻接表构建用户关系网络。
2. 节点重要性排序：
  - 实现度排序算法。
  - 实现基于投票策略的节点排序算法，包括投票能力的计算和更新机制。
3. 信息传播模拟：
  - 基于 SIR 模型，模拟信息在网络中的传播过程。
  - 记录不同种子节点组合下的感染规模，作为评价标准。
4. 结果对比与分析：
  - 对比度排序策略和基于投票策略在感染规模上的差异。
  - 分析不同初始种子节点数量和感染率对信息传播效果的影响。

## 六、实验器材（设备、元器件）：

IDE: IntelliJ IDEA 2024.1

Java: 17.0.11 2024-04-16 LTS

OS: Windows 10

## 七、实验步骤：

1. 根据选择的数据集构建 `GraphReader` 类，使用 `Scanner` 读取文件，支持读取节点与边的数量，通过邻接表构建图
2. 构建 `PageRank` 类，在初始化过程中读取节点数 `N`，以 `int[]` 数组存储各个节点的 ID 与出度（故而一个节点还有一个属性：`idx` 下标，用于检索其 ID 与出度），用 `HashMap` 构建两张邻接表：二者的键都是节点的 ID，其中，`graphOut` 的值为一个 `List<Integer>`，存储其指向的节点的 ID；`graphIn` 的值为一个 `int[]`，存储指向此节点的节点的 `idx` 下标，这是为了方便在计算 PR 时快速获得投票节点。对于出度为 0 的节点，则令其指向其他所有节点，以分摊其 PR 值，防止“吞噬”PR

3. 采用下面的公式循环计算各个节点的 PR（默认循环 500 次），设置  $d = 0.85$

$$PR(A) = \frac{1-d}{N} + d \left( \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \dots \right).$$

4. 建立记录 PageRankResult 以存储，处理 PageRank 的结果

5. 建立 SIRModel 类，构造函数的参数为 `Map<Integer, int[]> graph`，但与前面的 `graphIn` 不同的是，`int[]` 存储的是 ID 而不是 `idx`，在 `SIRModel` 里面没有下标的概念。在这里，我将每个节点抽象为一个 `Person`，每个 `Person` 都有其 `id` 和 `status`。`Status` 是一个 `enum`，有 `Susceptible`，`Infected`，`Recovered` 三种状态。在开始模拟时，所有人的状态都是 `Susceptible`，我们需要传入一个含有初始感染者 `id` 的数组（如果没有，则随机选择），将这些人的状态设置为 `Infected`。然后开始循环：遍历当前的所有已感染者，感染其邻居，邻居按照感染概率将其状态调整至 `Infected`；当遍历完后，这些感染者按照概率将状态按照恢复率（在此实验中必然恢复）调整至 `Recovered`，随后不再参与模型。接着开始下一轮的感染，直到没有感染者为止。最终统计所有曾经被感染的人，与总人数之比为感染率。

6. 在 `Main` 里面批量读取数据集，用 Page Rank，按度选择，随机选择三种策略选择重要节点，并将结果代入 SIR 模型进行测试，打印测试结果

## 八、实验数据及结果分析：

我收集了以下数据：

- <https://snap.stanford.edu/data/ca-AstroPh.html>
- <https://snap.stanford.edu/data/ca-CondMat.html>
- <https://snap.stanford.edu/data/ca-GrQc.html>
- <https://snap.stanford.edu/data/ca-HepPh.html>
- <https://snap.stanford.edu/data/ca-HepTh.html>

通过程序分析，得到了这五个数据集 PR 最大的 15 个节点，以及基于 SIR 模型，各个选择策略（Page Rank，最大入度，随机选择）的测试结果：

在这个简易的 SIR 模型上，以感染率 0.4，恢复率 1，初始感染人数为 11 模拟时：

Test test\CA-HepTh.txt

Page Rank Result: Largest 15 PR:

	ID	PR	In Degree
1	1441	0.0008750	65
2	19615	0.0007829	60
3	44262	0.0007756	51
4	23420	0.0007628	53
5	30744	0.0007615	56
6	63113	0.0006697	59
7	63697	0.0006433	38
8	14017	0.0006329	39
9	14642	0.0006328	38
10	16164	0.0006091	54
11	40517	0.0005963	49
12	59077	0.0005910	53
13	30160	0.0005821	49
14	13648	0.0005785	50
15	61742	0.0005771	50

Initializing infected [1441, 14017, 14642, 16164, 19615, 23420, 30744, 40517, 44262, 63113, 63697]

Initializing infected [1441, 13648, 16164, 19615, 23420, 30744, 44262, 48973, 59077, 61742, 63113]

Initializing infected [1049, 4885, 8660, 14481, 19177, 50213, 51152, 56264, 64829, 66996, 67269]

Test Page Rank, the infection ratio is: 0.582262

Test Largest Degree, the infection ratio is: 0.581756

Test Randomly Select, the infection ratio is: 0.575580

换了几种参数，除了感染率有明显变化外，发现三者的感染率不相上下，无法区分。得出的结论是：此模型的模拟思路无法体现初始感染者选择的差异

## 九、实验结论：

在本次针对复杂网络中重要节点组挖掘的实验研究中，我们采用了度排序策略、基于投票策略的 PageRank 算法以及随机选择策略来选取种子节点，并通过 SIR 模型模拟信息传播过程，我得出了如下结论：

1. PageRank 算法作为一种网页排序算法，在挖掘重要节点组方面也有良好的应用。比起单一的按度选择重要节点，这种方法基于投票进行排序，能够更好地刻画节点的影响力
2. 在简易的 SIR 模型中，感染率与恢复率对感染结果的影响过大，导致了初始选择节点对结果的重要性影响变小，需要更多参考现实中地情况：比如加入医院，潜伏期，口罩等一系列抽象

## 十、总结及心得体会：

PageRank 算法通过链接关系评估网页重要性，其创新性和简洁性对搜索引擎产生了深远影响，尽管存在局限性，但其迭代计算方式和广泛的应用领域证明了其强大的适用性和历史地位。

## 十一、对本实验过程及方法、手段的改进建议：

1. 可以使用 NDlib 提供的 SIR 模拟，使用 NetworkX 网络格式导入关系图，利用 MConfig 设置模型初始参数，以提供可视化模拟
2. 也可以尝试将挖掘出来的重要节点剔除，然后随机选择初始感染者，再进行 SIR 模型地模拟
3. 亦可以使用已知关键节点的网络，进行基准测试

报告评分：

指导教师签字：

# 附录：源代码

## PageRank.java

```
import java.util.*;

public class PageRank {
    private static final double d = 0.85;
    private final int N;          // number of nodes
    private final Map<Integer, List<Integer>> graphOut; // List: the id of next node
    private final Map<Integer, int[]> graphIn;    // List: the index of prev node
    private final int[] ids;      // id of nodes
    private final int[] Ls;       // out degree of nodes
    private static final int DEFALUT_RANK_TURNS = 500;

    public PageRank(String graphPath) {
        var gr = new GraphReader(graphPath);
        graphOut = gr.read(GraphReader.OUT);
        N = gr.readNumberOfNodes();
        ids = gr.getSortedIds();
        Ls = new int[N];

        List<Integer> outDegreeZero = new ArrayList<>();
        for (int i = 0; i < N; i++) {
            Ls[i] = L(i);
            assert Ls[i] > 0;
            if (Ls[i] == 0) {
                outDegreeZero.add(i);
                Ls[i] = N - 1;
            }
        }

        graphIn = new HashMap<>();
        var graphInOrigin = gr.read(GraphReader.IN);
        for (var entry : graphInOrigin.entrySet()) {
            var idList = entry.getValue();
            idList.addAll(outDegreeZero);
            int[] idxList = idList.stream().
                mapToInt(id -> Arrays.binarySearch(ids, id)).toArray();
            graphIn.put(entry.getKey(), idxList);
        }
    }

    private boolean checkIdx(int idx) {
        return idx >= 0 && idx < N;
    }

    /**
     * get out degree of a node
     */
}
```

```

* @param idx index of the node, ids[idx] = id of the node
* @return out degree of the node, -1 if id is illegal
*/
private int L(int idx) {
    var nodeList = graphOut.getDefault(ids[idx], null);
    return nodeList == null ? 0 : nodeList.size();
}

/**
* compute part of a node's new PR
* the sum of other node's PR dividing its out-degree which points to the node
* @param PR a list of the PR of nodes
* @param idx index of the node, which need a new PR
* @return the result, -1 if id is illegal
*/
private double sumOfAveragedPRVoteTo(double[] PR, int idx) {
    if (!checkIdx(idx))
        return -1;
    int id = ids[idx];
    var fromNodeList = graphIn.getDefault(id, null);
    if (fromNodeList == null)
        return 0;
    double ret = 0;
    for (int index : fromNodeList) {
        ret += PR[index] / Ls[index];
    }
    return ret;
}

private double nextPRi(double PR[], int idx) {
    double sum = sumOfAveragedPRVoteTo(PR, idx);
    return (1 - d) / N + d * sum;
}

private double[] originPR() {
    var ret = new double[N];
    Arrays.fill(ret, 1 / N);
    return ret;
}

private double[] nextPRList(double curPR[]) {
    var nextPR = new double[N];
    for (int i = 0; i < N; i++) {
        nextPR[i] = nextPRi(curPR, i);
    }
    return nextPR;
}

public RankResult rank(int n) {
    double[] PR = originPR();
    while ((n--) != 0) {
        PR = nextPRList(PR);
    }
}

```



```

Integer[] indices = new Integer[N];
for (int i = 0; i < indices.length; i++) {
    indices[i] = i;
}
final var sortedPR = PR;
Arrays.sort(indices, Comparator.comparingDouble(i -> sortedPR[i]));
Arrays.sort(sortedPR);

for (int i = 0; i < indices.length; i++)
    indices[i] = ids[indices[i]];

Map<Integer, int[]> copyGraphIn = new HashMap<>();
for (int id : graphIn.keySet()) {
    var idxList = graphIn.get(id);
    int len = idxList.length;
    var idList = new int[len];
    for (int i = 0; i < len; i++)
        idList[i] = ids[idxList[i]];
    copyGraphIn.put(id, idList);
}

return new RankResult(sortedPR, indices, copyGraphIn);
}

public RankResult rank() { return rank(DEFALUT_RANK_TURNS); }

/**
 * compare infectiousness between pagerank and random select
 * @return a double array containing the result of stimulation, the first is
 * pagerank's. the second is random selection's;
 */
public double[] testSIR() {
    var rankResult = rank();
    return rankResult.testSIR();
}
}

```

## RankResult.java

```

import java.util.Arrays;
import java.util.Comparator;
import java.util.Map;

public record RankResult(double[] sortedPR, Integer[] sortedIds, Map<Integer, int[]>
graphIn) {

```

```

public static final int PAGERANK = 0;
public static final int DEGREE = 1;
public static final int RANDOM = 2;

public String toString() {
    var sb = new StringBuilder();
    sb.append("    \t\tID \t\tPR    \t\tIn Degree\n");
    int len = sortedIds.length;
    for (int i = len - 1; i >= 0; i--) {
        int id = sortedIds[i];
        sb.append(String.format("%4d\t\t%4d\t\t%.7f\t\t%d\n", len - i, id,
sortedPR[i], graphIn.get(id).length));
    }
    return sb.toString();
}

public RankResult getLargestNPage(int n) {
    int len = sortedIds.length;
    return new RankResult(
        Arrays.copyOfRange(sortedPR, len - n, len),
        Arrays.copyOfRange(sortedIds, len - n, len),    // immutable
        graphIn
    );
}

public double[] getPR() {
    return sortedPR;
}

public Integer[] getIds() {
    return sortedIds;
}

/**
 * compare infectiousness between pagerank and random select
 * @return a double array containing the result of stimulation, the first is
pagerank's. the second is random selection's;
 */
public double[] testSIR() {
    var ret = new double[3];
    var sir = new SIRModel(graphIn);
    int n = SIRModel.DEFAULT_INITIAL_INFECTED_NUMBER;
    var selectPR = getLargestNPage(n).getIds();
    var selectDegree = graphIn.entrySet().stream()
        .sorted(Map.Entry.<Integer, int[]>comparingByValue(new
Comparator<int[]>() {
            @Override
            public int compare(int[] v1, int[] v2) {
                return v2.length - v1.length;
            }
        }))
        .limit(n)
        .map(Map.Entry::getKey)

```

```

        .toArray(Integer[]::new);
        ret[PAGERANK] = sir.stimulate(selectPR);
        ret[DEGREE] = sir.stimulate(selectDegree);
        ret[RANDOM] = sir.stimulate();
        return ret;
    }
}

```

## SIRModel.java

```

import java.util.*;

public class SIRModel {
    private final Map<Integer, int[]> graph;
    private final int N;
    private int nInfected = 0;
    private final Map<Integer, Person> persons;
    private Map<Integer, Person> theInfected;
    private enum Status {Susceptible, Infected, Recovered};
    public static final double DEFAULT_BETA = 0.4;
    public static final double DEFAULT_GAMMA = 1;
    public static final int DEFAULT_INITIAL_INFECTED_NUMBER = 11;

    public SIRModel(Map<Integer, int[]> graph) {
        this.graph = graph;
        N = graph.size();
        persons = new HashMap<>(N);
        for (int id : graph.keySet())
            persons.put(id, new Person(id));
    }

    private void initializeInfected(Integer[] initialInfectedId) {
        Arrays.sort(initialInfectedId);
        System.out.println("Initializing infected " +
Arrays.toString(initialInfectedId));
        nInfected = 0;
        theInfected = new HashMap<>(initialInfectedId.length * 2);
        for (int id : initialInfectedId) {
            var p = persons.get(id);
            p.infected();
        }
    }

    /**
     * Stimulate SIR Module, return the prevalence of epidemic
     * @param initialInfectedId the IDs of initial infection
     * @param infectP probability of being infected

```

```

    * @param recoverP probability of recovery
    * @return The proportion of people who have been infected with the disease
    */
    public double stimulate(Integer[] initialInfectedId,
                           double infectP, double recoverP)
    {
        for (var p : persons.values()) {
            p.formatted();
        }
        initializeInfected(initialInfectedId);
        while (!theInfected.isEmpty()) {
//            System.out.println(theInfected.size() + " " +
Arrays.toString(theInfected.keySet().toArray(Integer[]::new)));
            var curInfected = theInfected.values().toArray(Person[]::new);
            for (var p : curInfected) {
                var neighborIds = graph.get(p.getId());
                for (var neighborId : neighborIds) {
                    var neighbor = persons.get(neighborId);
                    neighbor.visited(infectP);
                }
                p.heal(recoverP);
            }
        }
//        System.out.println("nInfected: " + nInfected);
        return 1.0 * nInfected / N;
    }

    public double stimulate(Integer[] initialInfectedId) {
        return stimulate(initialInfectedId,
            DEFAULT_BETA, DEFAULT_GAMMA);
    }

    public double stimulate() {
        var allIds = set2Array(graph.keySet());
        return stimulate(
            selectRandomly(allIds, DEFAULT_INITIAL_INFECTED_NUMBER),
            DEFAULT_BETA,
            DEFAULT_GAMMA
        );
    }

    private Integer[] set2Array(Set<Integer> set) {
        return set.toArray(Integer[]::new);
    }

    private Integer[] selectRandomly(Integer[] ids, int n) {
        assert n <= ids.length;
        var listCopy = Arrays.asList(ids);
        Collections.shuffle(listCopy);
        return listCopy.stream().limit(n).toArray(Integer[]::new);
    }

    private class Person {

```

```

private Status status;
private final int id;

public Person(int id, Status status) {
    this.id = id;
    this.status = status;
}

public Person(int id) {
    this(id, Status.Susceptible);
}

public int getId() { return id; }

public void formatted() {
    status = Status.Susceptible;
}

public void recovered() {
    status = Status.Recovered;
    theInfected.remove(id);
}

public void infected() {
    status = Status.Infected;
    nInfected++;
    theInfected.put(id, this);
}

public void visited(double infectedP) {
    if (status == Status.Susceptible && chance(infectedP)) {
        infected();
    }
}

public void heal(double recoverP) {
    if (chance(recoverP)) {
        recovered();
    }
}

/**
 * Returns true with the specified probability
 * @param p the probability
 * @return true in p, false in (1 - p)
 */
private boolean chance(double p) {
    return Math.random() < p;
}
}

```

# Main.java

```
import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<String> filePaths = getFilePaths("test");
        // testFile("test\\CA-HepTh.txt");
        for (String filePath : filePaths) {
            testFile(filePath);
        }
    }

    private static void testFile(String filePath) {
        var pr = new PageRank(filePath);
        var pageRankResult = pr.rank();
        System.out.println("Test " + filePath);
        System.out.println("Page Rank Result: Largest 15 PR:");
        System.out.println(pageRankResult.getLargestNPage(15));
        var testResult = pr.testSIR();
        printTest("Page Rank", testResult[RankResult.PAGERANK]);
        printTest("Largest Degree", testResult[RankResult.DEGREE]);
        printTest("Randomly Select", testResult[RankResult.RANDOM]);
        System.out.println("\n-----");
    }

    private static void printTest(String test, double res) {
        System.out.printf("Test %s, the infection ratio is: %f\n", test, res);
    }

    private static List<String> getFilePaths(String directoryPath) {
        List<String> filePaths = new ArrayList<>();
        File directory = new File(directoryPath);
        if (directory.exists() && directory.isDirectory()) {
            File[] files = directory.listFiles();
            if (files != null) {
                for (File file : files) {
                    if (file.isFile()) {
                        filePaths.add(file.getAbsolutePath());
                    }
                }
            }
        }
        return filePaths;
    }
}
```

```
}
```

## GraphReader.java

```
import java.io.FileNotFoundException;
import java.util.*;
import java.io.File;

public class GraphReader {
    private final File file;
    private final int nodes;
    private final int edges;
    public static final boolean OUT = true;
    public static final boolean IN = false;

    public GraphReader(String filePath) {
        file = new File(filePath);
        assert file.exists();
        nodes = readNumberOf("Nodes:");
        edges = readNumberOf("Edges:");
    }

    private Scanner getScanner() {
        Scanner in;
        try {
            in = new Scanner(file);
        } catch (FileNotFoundException e) {
            in = null;
            e.printStackTrace();
            System.exit(-1);
        }
        return in;
    }

    public Map<Integer, List<Integer>> read(boolean mode) {
        Map<Integer, List<Integer>> graph = new HashMap<>(nodes);
        var in = getScanner();
        String lineRead;
        do {
            lineRead = in.nextLine();
        } while (lineRead.charAt(0) == '#');

        var firstEdge = lineRead.split("\\t");
        assert firstEdge.length == 2 : "Edge Should have two Nodes but read:\\n" +
lineRead;
        int fromId = Integer.parseInt(firstEdge[0]);
```

```

int toId = Integer.parseInt(firstEdge[1]);

do {
    if (mode == IN) {
        int tmp = fromId;
        fromId = toId;
        toId = tmp;
    }

    var nodeList = graph.getOrDefault(fromId, null);
    if (nodeList == null) {
        var newNodeList = new ArrayList<Integer>();
        newNodeList.add(toId);
        graph.put(fromId, newNodeList);
    } else {
        nodeList.add(toId);
    }

    if (in.hasNextInt()) {
        fromId = in.nextInt();
        toId = in.nextInt();
    }
    else break;
} while (true);

in.close();
return graph;
}

public int[] getSortedIds() {
    var set = new HashSet<Integer>(nodes);
    var in = getScanner();
    String lineRead;
    do {
        lineRead = in.nextLine();
    } while (lineRead.charAt(0) == '#');

    var sc = new Scanner(lineRead);
    set.add(sc.nextInt());
    set.add(sc.nextInt());
    sc.close();

    while (in.hasNextInt())
        set.add(in.nextInt());
    assert set.size() == nodes : "Failed to read nodes. Should read " + nodes + "
but get " + set.size();

    int[] ret = new int[nodes];
    int ptr = 0;
    for (int id : set) {
        ret[ptr++] = id;
    }
    Arrays.sort(ret);

```



```

        return ret;
    }

    private int readNumberOf(String s) {
        var sc = getScanner();
        String lineRead;
        while (sc.hasNextLine()) {
            lineRead = sc.nextLine();
            if (lineRead.charAt(0) != '#')
                break;
            var finder = new Scanner(lineRead);
            if (finder.findInLine(s) != null)
                return finder.nextInt();
        }
        return -1;
    }

    public int readNumberOfNodes() {
        return nodes;
    }

    public int readNumberOfEdges() {
        return edges;
    }
}

```