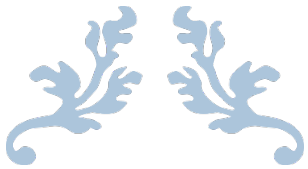


子图同构

Ullmann Algorithm with Refinement

DECEMBER 6, 2015

姜佳君

子图同构作业报告

姓名： 姜佳君 **学号：** 1501111332 **日期：** 2015.12.06

一、 课题介绍

子图同构在我们日常的生活中的应用越来越多，尤其是随着计算机科学和互联网的迅速发展，各种社交网络层出不穷。而在社交网络中不同用户之间存在各种错综复杂的关系，有时我们需要挖掘不同用户之间的各种关系网络。而对于相同关系网络的发掘，比如给定一个用户的部分关系用来检索特定用户等，属于子图同构的查询问题。但是，子图同构的查询算法属于 NP-hard 问题，目前为止并没有高效的算法来解决这个问题，其最坏的效果都是指数级的。如果针对特定的问题做一些优化，有时是可以降低时间复杂度的（理论上最坏时间复杂度依然是指数级，均摊法时间复杂度可以达到多项式时间复杂度）。Ullmann 算法就是采用了匹配矩阵的一种算法，结合剪枝降低时间复杂度，在实际的使用中可以达到一个比较好的效果。

二、 算法介绍

Ullmann算法，通过计算两个图（A和B）的匹配矩阵来判断其是否具有同构的特性。

匹配矩阵：$\mathrm{M}'[\mathrm{i}][\mathrm{j}] = 1$，表示图A中的第i个顶点匹配图B中的第j个顶点。由此，我们可知对于匹配矩阵M’的任意一行，存在最多一个1，即一个顶点只能对应一个顶点；同时，每一列也不能多于一个1.这样的一个匹配矩阵能够唯一的确定图A与图B的同构方式。

基于以上的结构，具体的算法如下描述。

Ullmann Algorithm：

输入：查询图A的边矩阵$MA_{m\times m}$，匹配图B的边矩阵$MB_{n\times n}$

输出：图A是否是图B的同构子图

第一步，计算匹配矩阵$M'_{m\times \mathrm{n}}$。$\mathrm{M}'[\mathrm{i}][\mathrm{j}] = 1$，当且仅当以下条件成立：

（1）在图A中第i个顶点的label与图B中的第j个顶点的label相同；

（2）在图A中第i个顶点的度数小于或等于图B中第j个顶点的度数。

第二步，对于匹配矩阵M’，每行保留一个1，并且保证每一列不能多于一个1。

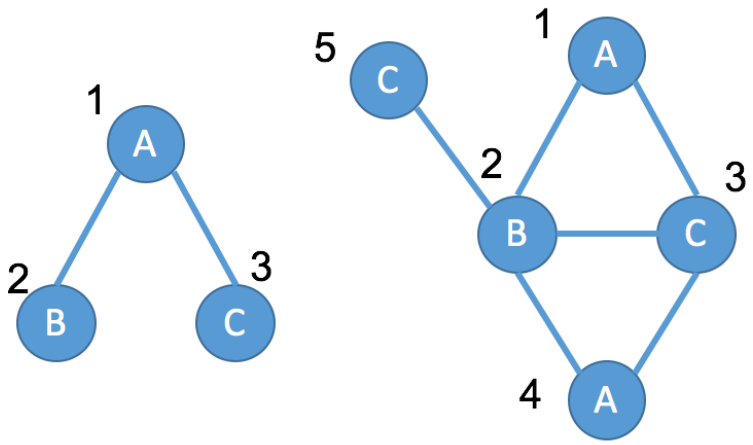
第三步，验证M’，是否满足子图同构的条件，如下：

$$\mathrm{MC} = \mathrm{M}'(\mathrm{M}' \cdot \mathrm{MB})^T \quad \forall_i \forall_j : (MA[i][j] = 1) \Longrightarrow (MC[i][j] = 1)$$

第四步，重复第二、三步，直到验证成功或者所有情况验证完。

说明：$MA_{m\times m}$ 与$MB_{n\times n}$是对称矩阵

上面算法是基本的算法描述，这样实现的时间复杂度还是比较高。在实际的计算过程中我们可以通过匹配顶点的邻居顶点的连接信息对搜索空间进行压缩，从而可以尽早的发现一些不满足条件的匹配。比如下面的例子：



根据上面的算法我们可以得到如下的匹配矩阵：

M	1	2	3	4	5
1	1	0	0	1	0
2	0	1	0	0	0
3	0	0	1	0	1

Can be removed

根据上图知图A中的顶点3C与图B中的顶点5C匹配，原因是其邻居顶点信息不符，在图A中3C的邻居顶点是1A，在图B中5C的邻居顶点是2B，所以表中的M[3][5]可以设置为0，即两个顶点不能进行匹配。总结出有以下的条件约束：

$$(\forall_{1\le x\le N} x)\quad (MA[i][x]=1)\Rightarrow \exists_{1\le y\le M} y\quad (M[x][y]\bullet MB[y][j]=1)$$

根据公式可以在上面的算法中的迭代过程中进行动态的检查，以降低计算时间复杂度。

三、 具体实现

根据上面的算法描述很容易实现算法代码，这里对算法不再详细描述，只针对用到的结构进行说明。

3.1 实验环境

操作系统：Windows 8.1 64bit

开发环境：Eclipse Mars

编程语言：Java

3.2 数据结构

1. 顶点结构

顶点主要包含顶点的编号、和顶点的 label 信息。

```
package structs;
/**
 * graph vertex class, include vertex id, vertex label and necessary
operations on them
 * @author Jiajun
 *
 */
public class Vertex {
    /*
     * vertex id
     */
    private int id;
    /*
     * vertex label
     */
    private String label;
    /**
     * constructor
     * @param id: vertex id
     * @param label: vertex label
     */
    public Vertex(int id, String label){
    }
    /**
     * get the vertex id of the vertex
     * @return
     */
    public int getId() {
    }
    /**
     * set the vertex id
     * @param id
     */
    public void setId(int id) {
    }
    /**
     * get vertex label
     * @return
     */
    public String getLabel() {
    }
    /**
     * set vertex label
     * @param label
     */
    public void setLabel(String label) {
    }

}
```

2. 边结构

边包含边的两个端点的编号以及边上的 label 信息。

```java
package structs;

/**
 * graph edge class, include the id of start and end vertex, also the
label of the edge is included.
 * @author Jiajun
 *
 */
public class Edge {
    /*
     * the start vertex id of the edge
     */
    private int start;
    /*
     * the end vertex id of the edge
     */
    private int end;
    /*
     * the label of the edge
     */
    private String label;
    /**
     * constructor
     * @param start: the id of the start vertex of the edge
     * @param end: the id of the end vertex of the edge
     * @param label: the label of the edge
     */
    public Edge(int start, int end, String label){
        this.start = start;
        this.end = end;
        this.label = label;
    }
    /**
     * get the id of the start vertex of the edge
     * @return
     */
    public int getStart() {
        return start;
    }
    /**
     * set the id of the start vertex of the edge
     * @param start
     */
    public void setStart(int start) {
        this.start = start;
    }
    /**
     * get the id of the end vertex of the edge
     * @return
     */
    public int getEnd() {
        return end;
    }
    /**
     * set the id of the end vertex of the edge
     * @param end
     */
    public void setEnd(int end) {
```

```
        this.end = end;
    }
    /**
     * get the label of the edge
     * @return
     */
    public String getLabel() {
        return label;
    }
    /**
     * set the label of the edge
     * @param label
     */
    public void setLabel(String label) {
        this.label = label;
    }
    /**
     * judging the given vertexes' id of start and end whether equal to
this's (with no directions)
     * TRUE if the start id equals to this start id and the end id
equals to this end id
     * TRUE if the start id equals to this end id and the end id equals
to this start id
     * @param start
     * @param end
     * @return
     */
    public boolean isEqual(int start, int end){
        if((this.start == start && this.end == end) || (this.end ==
start && this.start == end)){
            return true;
        }
        return false;
    }
}
```

3. 图结构

图是结构主要是包含图的所有顶点信息、边的信息和图的编号信息，以及该图的边的矩阵信息。

```
package structs;

import java.util.List;
/**
 * the graph class, include the graph index, the list of vertex, the
list of edge
 * and the matrix of the edge with no directions, in other words, if
there exist
 * an edge from the a(vertex) to b(vertex), there will be an edge from
b to a added
 * into the matrix, it is to say the matrix is symmetrical
 * @author Jiajun
 *
 */
public class Graph {
    /*
     * the id of the graph
```

```
     */
    private int id;
    /*
     * the list of the graph's vertexes
     */
    private List<Vertex> verList;
    /*
     * the list of the graph's edges
     */
    private List<Edge> edgeList;
    /*
     * the matrix of the edge, which is symmetrical
     */
    private int[][] matrix;
    /**
     * constructor
     * @param id: the index of the graph
     * @param vList: the list of the graph's vertexes
     * @param eList: the list of the graph's edges
     */
    public Graph(int id, List<Vertex> vList, List<Edge> eList){
        this.id = id;
        this.verList = vList;
        this.edgeList = eList;
        computeMatrix();
    }
    /**
     * compute the matrix of the edges
     */
    private void computeMatrix(){
        int v = verList.size();
        matrix = new int[v][v];
        for(Edge e : edgeList){
            int start = e.getStart();
            int end = e.getEnd();
            //NOTE: the matrix is symmetrical
            matrix[start][end] = 1;
            matrix[end][start] = 1;
        }
    }
    /**
     * get the number of the vertexes
     * @return
     */
    public int getVertexCount(){
        return this.verList.size();
    }
    /**
     * get the number of the edges
     * @return
     */
    public int getEdgeCount(){
        return this.edgeList.size();
    }
    /**
     * get the vertexes of the graph
     * @return
     */
```

```
    public List<Vertex> getVerList() {
        return verList;
    }
    /**
     * set the vertexes of the graph
     * @param verList
     */
    public void setVerList(List<Vertex> verList) {
        this.verList = verList;
    }
    /**
     * get the edges of the graph
     * @return
     */
    public List<Edge> getEdgeList() {
        return edgeList;
    }
    /**
     * get the id of the graph
     * @return
     */
    public int getId(){
        return this.id;
    }
    /**
     * set the edge list of the graph
     * @param edgeList
     */
    public void setEdgeList(List<Edge> edgeList) {
        this.edgeList = edgeList;
    }
    /**
     * get the edge matrix of the graph
     * @return
     */
    public int[][] getMatrix() {
        return matrix;
    }
    /**
     * get the degree of the given vertex's number
     * include in degrees and out degrees
     * @param number
     * @return
     */
    public int getDegree(int number){
        if(number > verList.size()){
            return 0;
        }
        int degree = 0;
        int v = verList.size();
        for(int i = 0; i < v; i++){
            degree += matrix[number][i];
        }
        return degree;
    }
    /**
     * get the label of the given edge's start and end vertexes' id
     * @param start
```

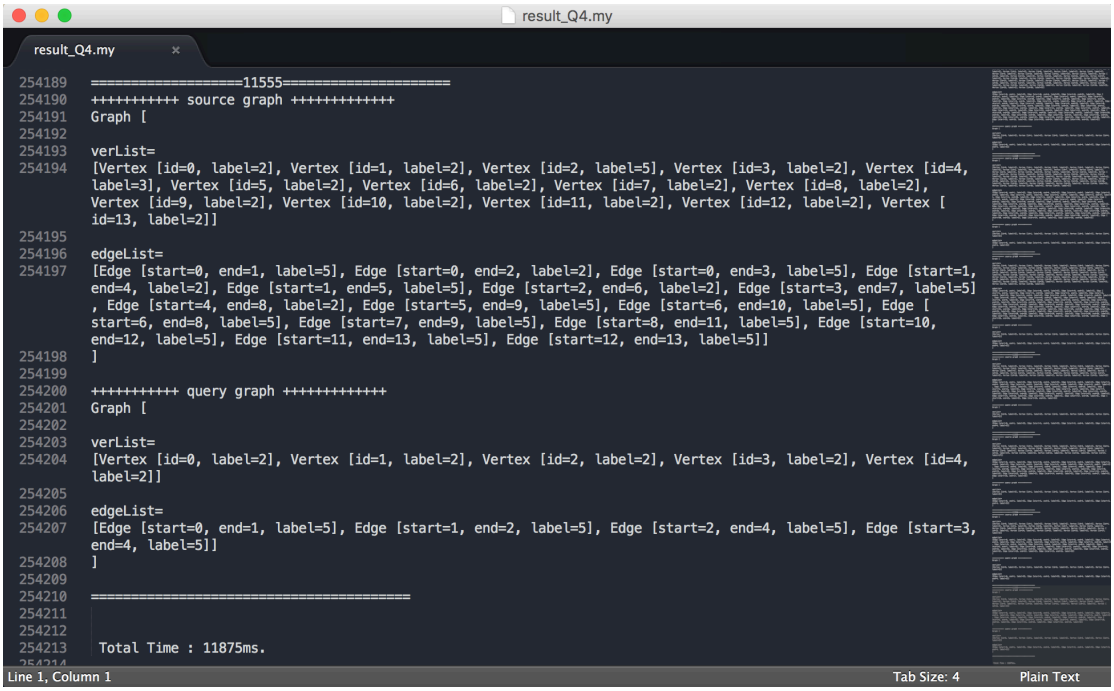
```
     * @param end
     * @return "NO" if there exits no such an edge
     */
    public String getEdgeLabel(int start, int end){
        String label = "NO";
        if(start < verList.size() && end < verList.size()){
            if(matrix[start][end] == 0){
                return label;
            }
            for(Edge e : edgeList){
                if(e.isEqual(start, end)){
                    label = e.getLabel();
                }
            }
        }

        return label;
    }
}
```

其他，具体实现见代码。

完整工程(https://github.com/xgdsmileboy/SubgraphIsomorphism)

四、实验

实验结果截图如下所示：（详细结果可访问我的 github）



说明：图中的 source graph 为 database 中的 graph，query graph 为查询的图。verList 是图中的所有顶点集合（id：顶点编号，label：顶点的标签），edgeList 是图中所有的边的集合（start、end：是边的两个端点，label：边的标签）。在匹配的过程中是顶点的标签和边的标签进行匹配，而不是编号进行匹配。

五、分析总结

分析上面的实验数据可知，算法运行正确，而且上述结果是对 10 个 query 子图进行检索，检索数据库大小为 10000 个图，共消耗时间为 11.875m，时间比较快，可见在使用了 Ullmann 算法并结合 refinement 方法，对子图同构是有很好的提速效果的。