目录

[自动化拓扑算法 2](#_Toc531204673)

[1、广度优先双端算法（场景AB） 2](#_Toc531204674)

[1.1 问题描述 2](#_Toc531204675)

[1.2解决方案 2](#_Toc531204676)

[1.3程序文件 3](#_Toc531204677)

[2、GN力导向布局算法（场景EF）： 3](#_Toc531204678)

[2.1问题描述 3](#_Toc531204679)

[2.2解决方案 4](#_Toc531204680)

[2.3程序文件 5](#_Toc531204681)

[3、非叶节点力导向布局算法（场景GH） 7](#_Toc531204682)

[3.1问题描述 7](#_Toc531204683)

[3.2解决方案 7](#_Toc531204684)

[4、力导向双端结合布局算法（场景I） 8](#_Toc531204685)

[4.1问题描述 8](#_Toc531204686)

[4.2解决方案 8](#_Toc531204687)

[4.3程序描述 8](#_Toc531204688)

# 自动化拓扑算法

## 1、广度优先双端算法（场景AB）

### 1.1 问题描述

对于这两个场景主要存在两个问题

（1）、主干节点连接分支节点的边之间会有交叉

（2）、在当分支节点与主干节点之间有环时没有进行处理

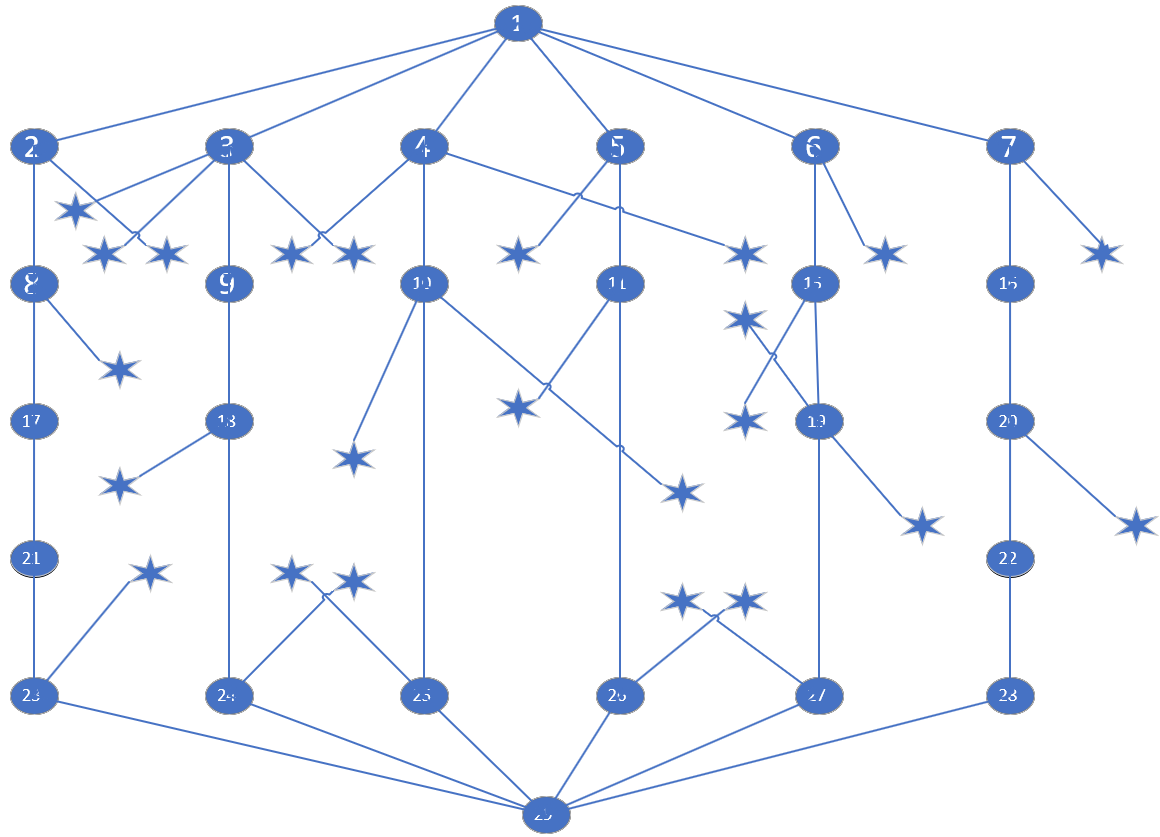


图1.1

### 1.2解决方案

（1）、寻找节点度数最高的两个点作为双端结构端点。

（2）、从端点出发遍历出所有主干节点，保存在数组中。

（3）、对每个主干进行广度优先遍历，对每一个与当前节点有边相连的节点排布在当前节点的圆周上。搜索过程，排布位置的分配如图2。



图2

### 1.3程序文件

对应文件为DoubleSide.java:

表1-1

|  |  |
| --- | --- |
| 函数名 | 功能 |
| public void summary() | //入口函数 |
| public void getMax() | //得到最大的度数的两个点 |
| public void DFSS() | //深度优先搜索的开始函数 |
| public void DFS(int v) | //深度优先搜索得到主干 |
| public void sort() | //对各个主干按长度进行排序 |
| public void process() | //主干节点坐标处理 |
| public void Unside() | //遍历各个主干的节点，对非主干节点进行排布 |
| public void init() | //初始化visit数组 |

## 2、GN力导向布局算法（场景EF）：

### 2.1问题描述

场景EF因为边的数目较多，力导向算法运行之后导致图形混乱，尤其当图标很大时，节点重叠和边的混乱交叉非常严重。在此基础上引入社团分析算法，结合改进的力导向算法进行处理。

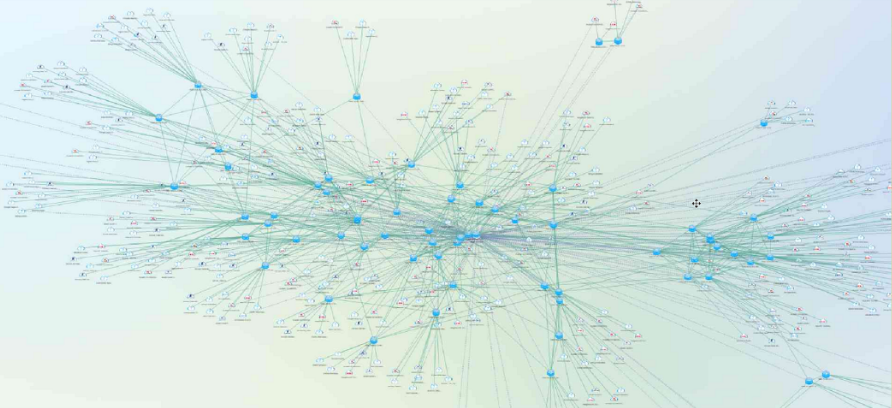


图2.1

### 2.2解决方案

在该场景中应用了两种GN算法。分别是传统GN算法和快速GN算法。在程序开始的版本里面传统算法时间复杂度较高，所以后面优化更改为快速GN算法。

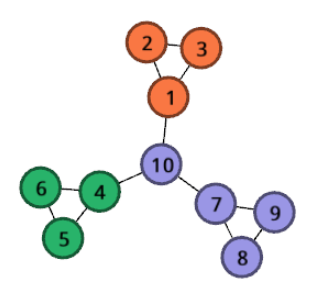
GN算法中应用到两个概念，边介数，模块度Q

1、边介数：网络中经过每条边的最短路径的数目（原始程序求解方法：Floyd最短路径算法）

2、模块度Q：

社团i内部总边数，为图的总边数，,社团i的点个数。

例如下面的网络 -- 10个节点，12条边，被分成3个社区：





一、传统GN 算法步骤：

1）计算网络中所有边的介数

2）找到介数最大的边并将它从网络中移除

3）重复步骤 2)，直到每个节点就是一个退化社团为止。

对于场景E，执行时间为3分钟

二、快速GN 算法步骤：

1）初始化所有点为单独社团

2）计算任意合并两个社团时其Q值增量，寻找合并后增量最大的两个社团进行合并

3）重复步骤 2)，直到所有点合并为一个社团为止。

采用贪心的方式，对于场景E,执行时间为2秒

社团i与社团j之间的边数，社团i与其他社团的边数。（计算时间o(1)）

在传统力导向算法中，通过在节点间斥力和节点之间的拉力寻找平衡。其中包括斥力常数拉力常数。

加入社团分析之后对传统力导向算法进行了改进。将原本的所有点之间的拉力斥力常数相同改为：同一社团之间拉力大斥力小，不同社团之间斥力大拉力小，并在图中寻找节点度数最大的两个点将其作为中心并标记为不属于任何社团。

### 2.3程序文件

快速GN算法为 FastGN.java

传统GN算法为GN.java

GN算法对应的力导向算法文件ForceDirectG.java

**GN.java程序入口public void G(Graph g) 函数。**

表2-1 GN.java函数介绍

|  |  |
| --- | --- |
| 函数名 | 功能 |
| void G(Graph g) | //GN算法入口 |
| int BFSS(Graph g) | //BFSS以及BFS用于GN算法执行过程中遍历连通分量以及标记每个点属于哪一个连通分量tag。 |
| double GetQ() | //计算当前情况下图的Q值。 |
| void InitFloyd() | //对Floyd算法将要用到的FloydA[][],FloydPath[][]矩阵进行初始化。 |
| void Floyd(Graph g) | //Floyd 算法 |
| void GetGFloyd() | //利用FloydPath矩阵，遍历任意两点间的最短路径 |
| void DelEdge(Graph g,double Q) | //找到G值最大的边并删除的算法，并将删除的边和删除边之前的Q值保存在RecordEdge数组中。 |
| void GetPath(int x,int y) | //遍历两点间的最短路径的同时，经过的边都将G值加1。 |
| void PreProcessForce() | //对GN算法的最后处理，找到最大Q值并对每个点标记社团 |

**FastGN.java程序入口public void Fast() 函数。**

表2-2 FastGN.java函数介绍

|  |  |
| --- | --- |
| public int Fast() | //程序入口 |
| public void converge(int a,int b) | //EdgeBetweenConn矩阵和ConnA数组进行更新 |
| public double IncreaseQ(int a,int b) | //计算Q值增量 |
| public double GetQ() | //计算Q值 |

**ForceDirectG.java程序入口public void direct() 函数。**

表2-3 ForceDirectG.java函数介绍

|  |  |
| --- | --- |
| 函数名 | 功能 |
| public void diect() | //本程序入口 |
| public void updateReplusion() | //更新拉力 |
| public void updateSpring() | //更新斥力 |
| public void update() | //更新距离 |
| public void repeatCalc(int n) | //重复迭代 |
| public void getMax() | //得到最大的度数的两个点 |

## 3、非叶节点力导向布局算法（场景GH）

### 3.1问题描述

对于GH场景来说，叶节点多，非叶节点分布稀疏。如果单纯使用力导向算法会导致叶节点和非叶节点区分困难，并且分布混乱。

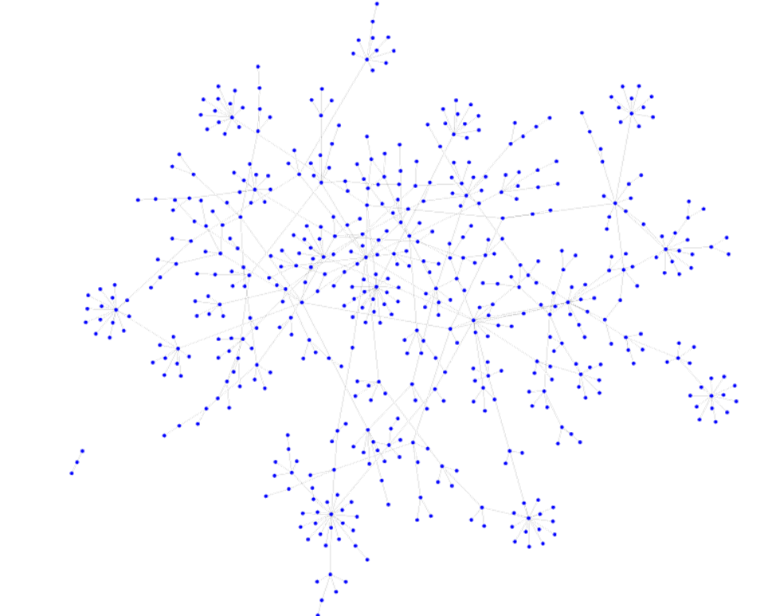


图3.1

### 3.2解决方案

所以处理方式是首先去掉叶节点，将非叶节点通过力导向算法进行排布，最后将所有节点的叶节点均匀的分布在非叶节点的圆周上。

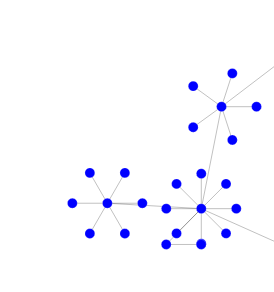
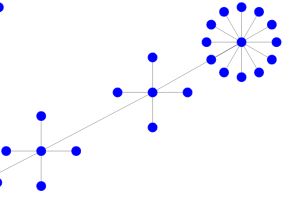
 

图3.2

## 4、力导向双端结合布局算法（场景I）

### 4.1问题描述

通过对图的观察可以发现大量的双端结构，但是双端结构的排布不能简单的应用顺序排列，这样无法达到区分层次的目的。所以首先应用力导向对度大于2的节点进行排列，排列后位置作为双端结构位置，具有较好的层次性。

### 4.2解决方案

场景I中可以检测到多个双端布局，所以采用如下算法。

1）、首先对degree>2的节点进行配对，作为双端布局的端点。

2）、应用力导向对所有degree>2的节点进行排布。

3）、在degree>2的节点位置进行双端布局。

4）、最后对未排布节点统一广度优先排布。

### 4.3程序描述

FDofNewDoubleSide.java程序入口public void diect() 函数。

NewDoubleSide.java程序入口public void start()函数

|  |  |
| --- | --- |
| 函数名 | 功能 |
| public void start() | //程序入口 |
| public int DFSS() | //深度优先搜索保存连通分量，并为每个连通分量分配位置 |
| public void searchAnother(int a,int pre) | //从二度节点a搜索另一端的端点 |
| public int find(int a) | //查找双端的另一端 |
| public void oppo() | //保存每个双端的端点 |
| public void subDFS(int v,int nodei,int nodej) | //搜索双端结构的主干 |
| public void subDFSS(int x,int y) | //搜索出双端结构的每一个主干 |
| public void sort() | //对双端结构的主干数组sub排序 |
| public void draw(Terminal t,double x,double y) | //为双端中的每个主干分配坐标，其中参数x,y是双端上端点的坐标 |
| public void dispose() | //为每个双端结构的上端点分配坐标，其实就是双端结构的位置 |
| public void BFS(int v) | //对所有不属于任何双端结构的点安装广度优先搜索分配坐标 |