图论与网络模型

周吕文

2015 年 04 月 15 日

引言

“图 (graph)” 这一名词是 Sylvester 在于 1878 年发表在自然上的一篇论文中提出的 [1]. 在数学和计算机科学中, 图论是研究图的一门科学. 图是由若干给定的点及连接两点的线所构成的图形, 这种图形通常用来描述某些事物之间的某种特定关系, 用点代表事物, 用连接两点的线表示相应两个事物间具有这种关系 [2]. 对于节点数比较多的图, 有时候也称为网络或者所谓的复杂网络. 我们的周围都是图和网络, 包括技术网络 (如因特网, 电网), 社会网络 (如社会图, 朋友网) 信息网络 (如万维网, 引用图等), 生物学网络 (如生化网, 神经网, 食物网) 等.

一般认为, 图论起源于 1736 年出版的 Euler 的关于柯尼斯堡七桥问题的论文 [3], 本节将在1.1介绍 “柯尼斯堡七桥问题 (Seven Bridges of Königsberg)” 更多细节. “柯尼斯堡七桥问题” 后来被推广为著名的欧拉路问题, 亦即一笔画问题. 欧拉关于柯尼斯堡七桥问题的论文与 Vandermonde 的一篇关于 “骑士周游问题 (knight problem)”[4] 的文章, 则是继承了 Leibniz 提出的 “位置分析”[5] 的方法. 欧拉提出的关于凸多边形顶点数, 棱数及面数之间的关系的 Euler 公式与图论有密切联系, 此后又被 Cauchy[6] 等人进一步研究推广, 成了拓扑学的起源. 1857 年, Hamilton 发明了 “环游世界游戏 (icosian game)”[7], 与此相关的则是另一个广为人知的图论问题 “哈密顿路径问题 (Hamiltonian path problem)”[8].

Euler 的论文发表后一个多世纪, Cayley 研究了在微分学中出现的一种数学分析的特殊形式, 而这最终将他引向对一种特殊的被称为 “树” 的图的研究 [9]. 由于有机化学中有许多树状结构的分子, 这些研究对于理论化学有着重要意义, 尤其是其中关于具有某一特定性质的图的计数问题. 除 Cayley 的成果外, Pólya[10] 也于 1935 至 1937 年发表了一些成果, 1959 年, De Bruijn[11] 做了一些推广. 这些研究成果奠定了图的计数理论的基础. Cayley 将他关于树的研究成果与当时有关化合物的研究联系起来, 而图论中有一部分术语正是来源于这种将数学与化学相联系的做法 [12].

1860 年之 1930 年间, Jordan, Kuratowski 和 Whitney 从之前独立于图论而发展的拓扑学中吸取大量内容进入图论, 而现代代数方法的使用更让图论与拓扑走上共同发展的道路. 图论中概率方法的引入, 尤其是 Paul Erdős 和 Alfréd Rényi 关于随机图连通的渐进概率的研究使得图论产生了新的分支随机图论 [2].

在本节中, 1.1和1.2中介绍两个经典的问题, 即柯尼斯堡七桥问题和地图染色问题, 让读者感受一下如何将现实中的问题转化为图论问题. 在1.3中介绍一下近些年中的数学建模比赛中涉及图论与网络模型的问题.

柯尼斯堡七桥问题 [13]

柯尼斯堡七桥问题是图论中的著名问题. 这个问题是基于一个现实生活中的事例: 当时东普鲁士柯尼斯堡 (今日俄罗斯加里宁格勒) 市区跨普列戈利亚河两岸, 河中心有两个小岛, 小岛与河的两岸有七条桥连接. 在所有桥都只能走一遍的前提下, 如何才能把这个地方所有的桥都走遍?

欧拉在 1735 年提出, 并没有方法能圆满解决这个问题. 他在第二年发表的论文 “柯尼斯堡的七桥” 中, 证明符合条件的走法并不存在, 也顺带提出并解决了一笔画问题. 这篇论文在圣彼得

堡科学院发表, 成为图论史上第一篇重要文献. 欧拉把实际问题抽象简化为平面上的点与线组合, 每一座桥视为一条线, 桥所连接的地区视为点, 如图1所示. 这样若从某点出发后最后再回到这点, 则这一点的线数必须是偶数, 这样的点称为偶顶点. 相对的, 连有奇数条线的点称为奇顶点. 欧拉论述了, 由于柯尼斯堡七桥问题中存在 4 个奇顶点, 它无法实现符合题意的遍历.

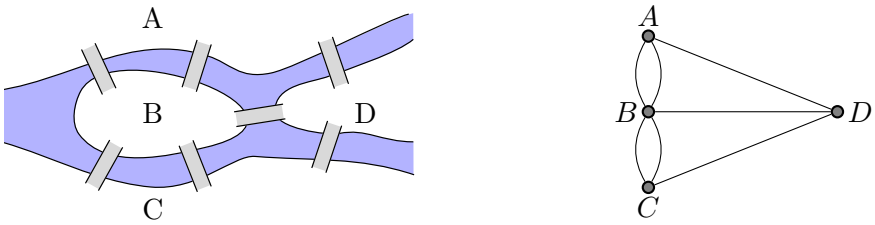


图 1: 柯尼斯堡七桥问题及图论表述

欧拉把问题的实质归于一笔画问题, 即判断一个图是否能够遍历完所有的边而没有重复, 而柯尼斯堡七桥问题则是一笔画问题的一个具体情境. 欧拉最后给出任意一种河　桥图能否全部走一次的判定法则, 从而解决了 “一笔画问题”. 对于一个给定的连通图, 如果存在两个以上 (不包括两个) 奇顶点, 那么满足要求的路线便不存在了, 且有 n 个奇顶点的图至少需要 $n/2$ 笔画出. 如果只有两个奇顶点, 则可从其中任何一地出发完成一笔画. 若所有点均为偶顶点, 则从任何一点出发, 所求的路线都能实现, 他还说明了该怎样快速找到所要求的路线.

地图染色问题 [14]

四色问题可谓是图论研究史上最著名也是产生成果最多的问题之一: “是否任何一幅画在平面上的地图都可以用四种颜色染色, 使得任意两个相邻的区域不同色?”. 这一问题最早于 1852 年由 Francis Guthrie 提出, 最早的文字记载则出现于 De Morgan 在同一年写给 Hamilton 的信上. 包括 Cayley, Kempe 等在内的许多人都曾给出过错误的证明. Tait, Heawood 及 Hadwiger 对此问题的研究与推广引发了对嵌入具有不同规格的曲面的图的着色问题的研究. 一百多年后, 四色问题仍未解决. 1969 年, Heinrich Heesch 发表了一个用计算机解决此问题的方法. 1976 年, Appel 和 Haken 借助计算机给出了一个证明, 此方法按某些性质将所有地图分为 1936 类并利用计算机一一验证了它们可以用四种颜色染色. 但此方法由于过于复杂, 在当时未被广泛接受.

四色问题的阐述可以转化为更为抽象的图论版本. 即将一个地图转化为图论中的一个平面无向图. 具体来说, 是将地图中的每一个国家用其内部的一个点代表, 作为一个顶点. 如果两个国家相邻, 就在两个顶点之间连一条线, 如图2所示. 这样得到的图必然是一个平面图 (不会有两条

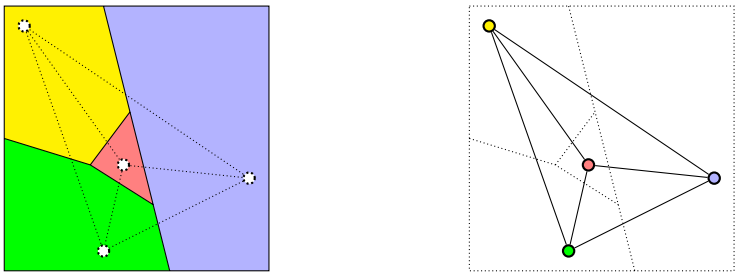


图 2: 四色问题及图论表述

边相交), 而与每个国家选取的代表点无关. 四色定理可以叙述为: 必然可以用四种颜色给平面图的顶点染色, 使得相连的顶点颜色不同.

数学建模竞赛中的应用情况

图论与网络模型是数学建模竞赛中最常用的方法之一. 在近些年来的美国大学生数学建模竞赛 (MCM) 和中国大学生数学建模竞赛 (CUMCM) 中, 出现了许多与图论和网络模型相关的赛题. 表1仅列出近五来年美国大学生数学建模竞赛中用到图论与网络模型的特等奖论文的不完全统计. 从表1中不难发现, 近些年来, 美国大学生数学建模竞赛每年都会出现至少一道与网络模型相关的赛题 [15]. 这提醒我们参赛者, 具体赛题我们无法预知, 但所涉及方法我们却是可以提前准备的.

表 1: 近五年 MCM 中涉及图论与网络模型方法的特等奖论文统计

年份题号	题目	特等奖论文数
2011 MCM-B	Repeater Coordination	4
2012 MCM-B	Modeling for Crime Busting	7
2013 ICM-C	Network Modeling of Earth's Health	5
2014 MCM-B	College Coaching Legends	1
2014 ICM-C	Using Networks to Measure Influence and Impact	6
2015 ICM-C	Managing Human Capital in Organizations	6

基本概念

图 (无向图) 和有向图

现实生活中, 许多问题可以抽象为若干点及连接两点的线所构成的图. 例如点可以表示人, 两点间连线表示两人为朋友关系; 再如用点表示城市, 两点间连线表示城市间是否有公路直接相连. 数学上, 我们把这种抽象出来的由点和边构成的图形称为图 (graph). 一个图 G[1]是由一个有序偶 $\big(V(G), E(G)\big)$ 及关联函数 (incidence function)φ_G 构成. 其中 $V(G)$ 是图 G 的顶点集或节点集, $V(G)$ 中的任意一个元素称为图 G 的一个顶点 (vertex) 或节点 (node); $E(G)$ 是不与 $V(G)$ 相交的边集. 如果 $e \in E(G)$ 是一条连接顶点 u 和 v 的边 (edge), 则有 $\varphi_G(e) = uv$. 我们称顶点 u 和 v 为边 e 的端点, 并称 u 与 v 相邻 (adjacent); 边 e 与 v, u 关联 (incident).

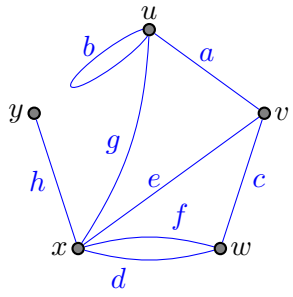


图 3: 无向图 G

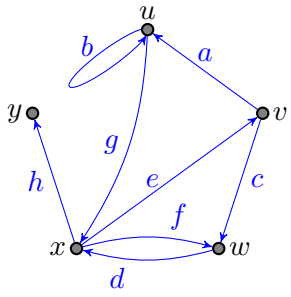


图 4: 有向图 D

通常, 图的一个顶点可以用平面上的一个点来表示, 边可以用平面上的线段来表示 (直线或曲线), 这些边仅在端点相交, 这样画出的平面图形称为图的图形表示. 这里我们给出一个无向图实例, 如图3所示, 无向图 $G = \big(V(G), E(G)\big)$, 其中 $V(G) = \{u, v, w, x, y\}$, $E(G) = \{a, b, c, d, e, f, g, h\}$. 关联函数 φ_G 的定义如下:

$$\varphi_G(a) = uv, \quad \varphi_G(b) = uu, \quad \varphi_G(c) = vw, \quad \varphi_G(d) = wx,$$
$$\varphi_G(e) = vx, \quad \varphi_G(f) = wx, \quad \varphi_G(g) = ux, \quad \varphi_G(h) = xy.$$

[1]若未指明为 "有向图 (directed graph)", 则通常 "图" 都指无向图 (undirected graph).

在图3, 像边 b 这样端点重合为一点的边称为**环 (loop)**. 反之, 端点不重合的边称为**连杆 (link)**. 如果有两个或多个连杆具有相同的两个端点, 称这样的边为**重边 (parallel edges)**. 如果一个图既没有环也没有重边, 则该图称为**简单图 (simple graph)**, 显然图3不是一个简单图, 它不仅存在一个环 b, 还存一对重边 f 和 d. 边上赋权的无向图称为**赋权无向图**. 如果一个图的顶点集和边集都是有限的, 这样的图称为**有限图 (finite graph)**. 图 G 的顶点数和边数分别用 $|V|$(或 $v(G)$) 和 $|E|$(或 $\varepsilon(G)$) 表示.

虽然很多问题都可以转化为无向图, 但仍有很多问题无向图的概念不足以表征. 比如在研究交通流问题时, 有些道路是单向的. 这时候我们就需要引进一种边是具有方向的图, 我们称这种图为**有向图 (directed graph)**. 一个有向图 D 是由一个有序偶 $\big(V(D), A(D)\big)$ 及关联函数 φ_D 构成. 其中 $V(D)$ 是图 D 的顶点集或节点集, $V(D)$ 中的任意一个元素称为图 D 的一个顶点或节点; $A(D)$ 是不与 $V(G)$ 相交的弧集. 如果 $a \in A(D)$ 是一条从顶点 u 到 v 的弧 (arc), 则有 $\varphi_D(a) = (u, v)$ 或 $\varphi_D(a) = uv$. 我们称顶点 u 为弧 a 的尾 (tail), v 为弧 a 的头 (head), 并称弧 a 为 u 的出弧 (outgoing arc), 为 v 的入弧 (incomming arc).

这里我们给出一个有向图实例, 如图4所示, 有向图 $D = \big(V(D), A(D)\big)$, 其中 $V(D) = \{u, v, w, x, y\}$, $A(D) = \{a, b, c, d, e, f, g, h\}$. 关联函数 φ_D 的定义如下:

$$\begin{aligned} &\varphi_D(a) = vu, && \varphi_D(b) = uu, && \varphi_D(c) = vw, && \varphi_D(d) = wx, \\ &\varphi_D(e) = xv, && \varphi_D(f) = xw, && \varphi_D(g) = ux, && \varphi_D(h) = xy. \end{aligned}$$

对应于每个有向图 D, 可以在相同顶点集上作一个图 G, 使得对于 D 的每条弧, G 有一条有相同端点的边与之相对应. 这个图称为 D 的基础图. 反之, 给定任意图 G, 对于它的每个边, 给其端点指定一个顺序, 从而确定一条弧, 由此得到一个有向图, 这样的有向图称为 G 的一个定向图. 图3和4中的两个图便是这样的一对 G, D.

子图

两个图 G 和 H, 如果 $V(H)$ 是 $V(G)$ 的子集且 $E(H)$ 是 $E(G)$ 的子集, 则称 H 是 G 的**子图 (subgraph)**. 如果图 G 和 H 不相等, 即 $V(H)$ 是 $V(G)$ 的真子集或 $E(H)$ 是 $E(G)$ 的真子集，则称 H 是 G 的真子图. 如果 H 是 G 的子图或者存在一个 G 的子图与 H 同构，则称 G 包含 H. 图2.2所示是图 G 及其去边和去顶点子图.

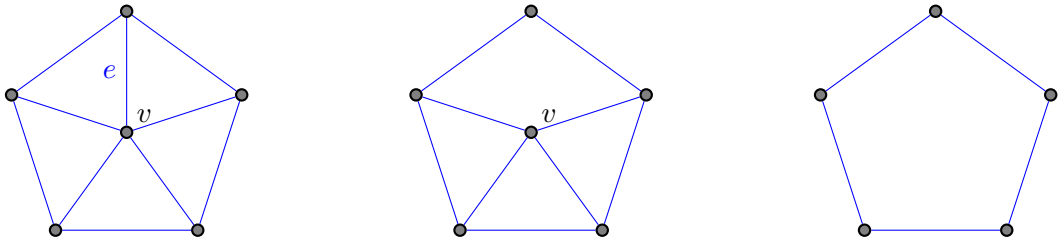


图 5: 图 G 及其去边和去顶点子图

如果图 G 的子图 H 满足 $V(H) = V(G)$, 即图 H 包含图 G 的所有顶点, 则称 H 是 G 的支撑子图或生成子图. 如果图 G 的子图 H 满足边 (u, v) 在图 H 中当且仅当边 (u, v) 在图 G 中, 即图 H 包含了图 G 中所有两个端点都在 $V(H)$ 中的边, 则称 H 是 G 的导出子图. 对于图的某个性质而言, 如果图 G 具有此性质而 G 的任一真子图都不具有此性质, 则称 G 是具有该性质的极小图. 类似地, 如果图 G 具有此性质而任一以 G 为真子图的图都不具有此性质, 则称 G 是具有该性质的极大图.

一些特殊的图

某些特定类型的图在图论中扮演了重要的角色. 每一对不同的顶点都有一条边相连的简单图称为**完全图 (complete graph)**, 任意两顶点都不存在边相连 (边集为空) 的图称为**空图 (empty graph)**. 如果一个图的顶点集可以分成两个子集 X 和 Y, 使得任意一条边的两个端点一个来自 X, 另一个来自 Y, 这样的图称为**二分图 (bipartite graph)**. 如果二分图是一个简单图并且所有 X 中的顶点都与所有 Y 中的顶点相邻, 则称这样的二分图为**完全二分图 (complete bipartite graph)**. 对于一个 $|X| = 1$ 或 $|Y| = 1$ 的完全二分图, 又称为**星图 (star)**. 图6-8显示的分别是完全图, 完全二分图和星图的实例.

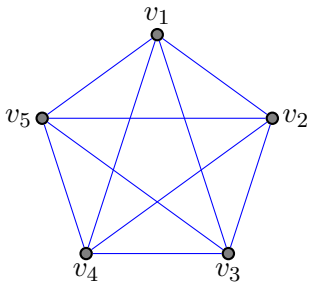


图 6: 完全图

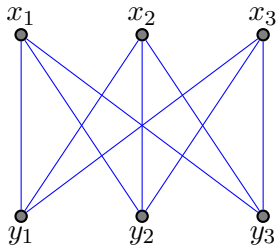


图 7: 完全二分图

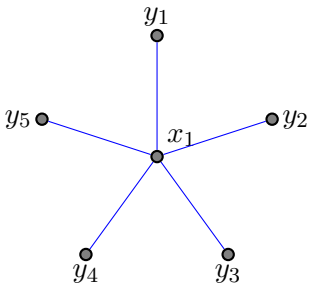


图 8: 星图

如果一个图中的顶点任意分成两个非空顶点集 X 和 Y, 都存在至少一条边的两顶点分别在 X 和 Y 中, 则称这样的图为**连通图 (connected graph)**, 反之则称为**不连通图 (disconnected graph)**. 图9和图10分别给出了一个连通图和不连通图的实例.

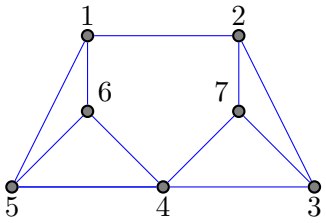


图 9: 连通图

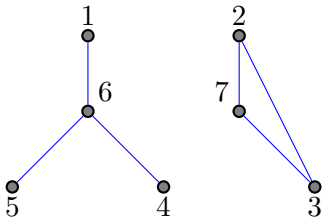


图 10: 不连通图

在图论中, 一条路也是一个简单图, 一条**路**是一个顶点序列, 使得从它的每个顶点有一条边到该序列中下一顶点. 一条道路可能是无穷的, 但有限道路有一个最先顶点, 称为起点, 和最后顶点, 称为末点. 两者都成为这条道路的端点. 路中其它顶点成为内点. 一个**圈**是起点与末点相同的路. 注意到一个圈中起点的选取是任意的. 只有一个顶点的圈是环, 只有两个顶点的圈是重边. 没有圈的连通图称为**树 (tree)**, 或者说, 树是任意两个顶点间存在唯一一条路径的图. 在树中, 仅与一条边关联 (度为 1) 的顶点称为叶子, 否则称为内点. 图11-13分别给出了一个路, 一个圈和一个树的实例.

图与网络的数据结构

一个图或网络的图形, 虽然能够很直观的表示出图或网络, 但这种方式却无法在计算机中描述图或网络. 为了在计算机上实现图和网络的相关算法, 我们必须有一种方法 (即数据结构) 在计算机上来描述图与网络. 一般来说, 算法的好坏与网络的具体表示方法, 以及中间结果的操作方案是有关系的. 计算机上用来描述图与网络的主要表示方法有关联矩阵表示法, 邻接矩阵表示法, 弧表表示法, 邻接表表示法和星形表示法. 这里我们介绍其中两种最为常用的表示方法: 邻接矩阵和关联矩阵表示法. 下面我们就无向图和有向图分别介绍关联矩阵和邻接矩阵表示法.

对于一个无向图 $G = (V, E)$, 假设有 $|V| = n$, $|E| = m$. 则图 G 的关联矩阵 $\mathbf{M}_G = (m_{ve})$ 是一个 $n \times m$ 的矩阵. 其中 $m_{ve} \in \{0, 1, 2\}$ 表示边 e 与顶点 v 关联的次数. 一个顶点的环被认为

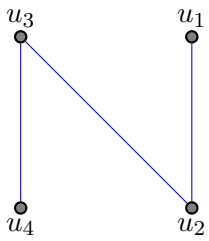


图 11: 路

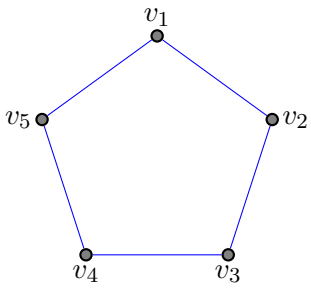


图 12: 圈

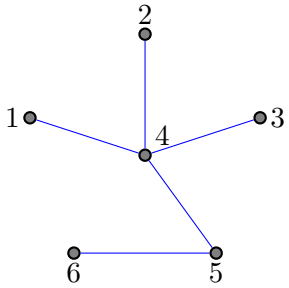
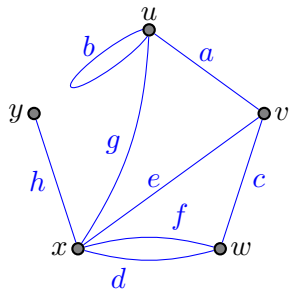


图 13: 树

与该顶点关联次数为 2. 图 G 的邻接矩阵 $\mathbf{A}_G = (a_{uv})$ 是一个 $n \times n$ 的矩阵. 其中 a_{uv} 表示连接顶点 u 和 v 间边的条数. 一个顶点的环被算为两条边. 显然, 对于无向图, 邻接矩阵 $\mathbf{A}_G$ 必然是一个对称阵. 图14是一个无向图 G 及其关联矩阵 $\mathbf{M}_G$ 和邻接矩阵 $\mathbf{A}_G$ 的实例.



G

	a	b	c	d	e	f	g	h
u	1	2	0	0	0	0	1	0
v	1	0	1	0	1	0	0	0
w	0	0	1	1	0	1	0	0
x	0	0	0	1	1	1	1	1
y	0	0	0	0	0	0	0	1

M

	u	v	w	x	y
u	2	1	0	1	0
v	1	0	1	1	0
w	0	1	0	2	0
x	1	1	2	0	1
y	0	0	0	1	0

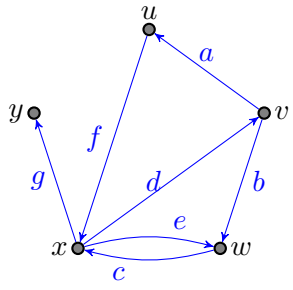
A

图 14: 无向图 G 及其关联矩阵 $\mathbf{M}_G$ 和邻接矩阵 $\mathbf{A}_G$

对于一个有向无圈图 $D = (V, A)$, 假设有 $|V| = n$, $|A| = m$. 则图 D 的关联矩阵 $\mathbf{M}_G = (m_{va})$ 是一个 $n \times m$ 的矩阵. 其中 $m_{va} \in \{0, 1, -1\}$ 表示弧 a 与顶点 v 关联的关系:

$$m_{va} = \begin{cases} 1 & \text{弧 } a \text{ 是一条连杆, 并且顶点 } v \text{ 是 } a \text{ 的尾} \\ -1 & \text{弧 } a \text{ 是一条连杆, 并且顶点 } v \text{ 是 } a \text{ 的头} \\ 0 & \text{其它} \end{cases}$$

图 D 的邻接矩阵 $\mathbf{A}_D = (a_{uv})$ 是一个 $n \times n$ 的矩阵. 其中 a_{uv} 表示是否存在从顶点 u 到 v 的弧. 即如果两节点之间有一条弧, 则邻接矩阵中对应的元素为 1; 否则为 0. 显然, 与无向图不同的是, 有向图的邻接矩阵一般情况并不对称. 图15是一个有向图 D 及其关联矩阵 $\mathbf{M}_D$ 和邻接矩阵 $\mathbf{A}_D$ 的实例.



D

	a	b	c	d	e	f	g
u	1	0	0	0	0	-1	0
v	-1	-1	0	1	0	0	0
w	0	1	-1	0	1	0	0
x	0	0	1	-1	-1	1	-1
y	0	0	0	0	0	0	1

M

	u	v	w	x	y
u	0	0	0	1	0
v	1	0	1	0	0
w	0	0	0	1	0
x	0	1	1	0	1
y	0	0	0	0	0

A

图 15: 有向图 D 及其关联矩阵 $\mathbf{M}_D$ 和邻接矩阵 $\mathbf{A}_D$

对于边上赋权的图或网络, 也可以用类似邻接矩阵的 $n \times n$ 矩阵表示. 此时一条弧所对应的元素不再是 1, 而是相应的权而已. 如果网络中每条弧赋有多种权, 则可以用多个矩阵表示这些权.

顶点的度和中心度

在图论中, 对于无向图 G, 顶点 $v \in V(G)$ 的**度 (degree)** 定义为: G 中与 v 关联的边数 (每个环算作两条边), 记作 $d_G(v)$ 或 $\deg(v)$. 如果 $d_G(v)$ 是奇数, 称 v 是奇顶点 (odd point); 如果 $d_G(v)$ 是偶数, 称 v 是偶顶点 (even point). 对于顶点的度, 有以下性质[2]

- 所有顶点的度之和等于两倍的边数, 即 $\sum_{v \in V} d(v) = 2|E|$.
- 任意一个图的奇顶点的个数是偶数.

对于有向图, 顶点的度又可细分为出度 (outdegree) 和入度 (indegree):

- 出度: 以某顶点为弧尾, 起始于该顶点的弧的数目称为该顶点的出度. 顶点 v 的出度记作 $d^-(v)$ 或 $\deg^-(v)$. 例如对于图15中的 D 图, 顶点 x 的出度为 $d^-(x) = 3$.
- 入度: 以某顶点为弧头, 终止于该顶点的弧的数目称为该顶点的入度. 顶点 v 的入度记作 $d^+(v)$ 或 $\deg^+(v)$. 例如对于图15中的 D 图, 顶点 x 的入度为 $d^+(x) = 2$.

对于有向图, 所有顶点的出度和必然等于所有顶点的入度和, 且等于弧数, 即 $\sum_{v \in V} d^-(v) = \sum_{v \in V} d^+(v) = |A|$.

在网络, 特别是社会网络的分析中, 经常用到中心度 (Centrality) 的概念来描述个人或者组织在社会网络中具有的权力, 或者说居于的中心地位, 对于信息在整个网络中如何传播, 以及传播效果都有十分重要的意义. 最常见的度量节点的中心度的方式有点度中心度, 接近中心度, 中间中心度和特征向量中心度. 下面我们给出这四种中心度的定义 [17, 18], 并以图16所示的社交网络加以说明:

- **点度中心度 (Degree Centrality)**: 点度中心度是最常见, 也是最简单的方法. 在无向图中, 用一个节点的度数来衡量点度中心度. 在有向图中, 可以用节点的入度表示点度中心度. 在很多情况下, 无向图中的边可以认为是双向的. 因此无论是无向图还是有向图, 顶点 v 的点度中心度 $C_D(v)$ 可以统一表示为:

  $$C_D(v) = d^+(v)$$

  我们称 $C_D(v)$ 为绝对点度中心度, 相应的标准化中心度为 $C_D(v)/(|V| - 1)$.

  在社会网络中, 一个用户与其他很多用户有直接联系, 该用户就处在中心地位. 即朋友越多, 越显示出来节点的重要性. 在图16中, 节点 Dave 的好友数最多, 有 6 个人, 所以他成为了点度中心度最高的节点. 真实的社交网络中, 点度中心度高的那些人一般都是公众人物, 有较大的知名度.

- **接近中心度 (Closeness Centrality)**: 在连通图中, 可以用顶点间的最短路径来衡量顶点间的距离. 定义顶点 v 与其它所有顶点的距离和的倒数为其接近中心度, 即

  $$C_C(v) = \frac{1}{\sum_{u \in V} d(u, v)}$$

  我们称 $C_C(v)$ 为绝对接近中心度, 相应的标准化中心度为 $C_C(v) \cdot (|V| - 1)$. 显然, 如果节点到图中其它节点的最短距离都很小，那么我们认为该节点的接近中心度较高. 这个定义

[2]对于这些性质, 我们不作证明, 有兴趣的同学参看专门的图论教材 [16].

其实比点度中心度从几何上更符合中心度的概念, 因为到其它节点的平均最短距离最小, 意味着这个节点从几何角度看是处于图的中心位置.

在社会网络中, 当用户越是离其他人近, 则在传播信息的过程中越是不依赖其他人. 因为一个非核心成员必须通过其它人才能传播信息, 容易受制于其它节点. 在图16中, Fanny 和 Garth 虽然好友数不如 Dave, 但他们到其它所有节点的最短距离是最小的. 直观上说, Dave 虽然好友数多, 但离图的右半部分的节点更加的远. 真实的社交网络中, 接近中心度高的节点一般扮演的是八婆的角色. 他们并不是明星, 但是乐于在不同的人群之间传递消息.

- **中间中心度 (Between Centrality)**: 定义顶点 v 作为其它两个节点间最短路径的中间点的频率为该顶点的中间中心度. 即

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

其中 σ_{st} 表示的是节点 s 和 t 之间的最短路径的数量, 而 $\sigma_{st}(v)$ 是最短路径中经过节点 v 的数量. 计算网络中任意两个节点的所有最短路径, 如果这些最短路径中有很多条都经过了某个节点, 那么就认为这个节点的中间中心度高.

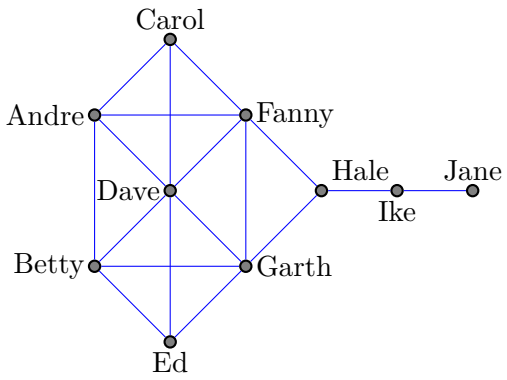
在社会网络中, 如果一个用户处在许多交往网络的路径上, 则认为此人处于重要地位, 因为该人具有控制他人交往的能力, 其他人的交往需要通过该人才能进行. 因而中间中心度测量的是用户对资源信息的控制程度. 在图16中, Hale 就是中间中心度最高的节点，因为 Ike 和 Jane 到其它节点的路径都需要经过 Hale.

- **特征向量中心度 (Eigenvector Centrality)**: 特征向量中心度是衡量网络上节点的影响力的重要指标. 一个节点的特征向量中心度等于与其相邻节点的中心度线性叠加. 特征向量中心度的思想是: 与高中心度节点相邻对自身中心度的提升贡献要比与低中心度节点相邻大.

$$C_E(v) = x_v = \frac{1}{\lambda} \sum_{u \in M(v)} x_u = \frac{1}{\lambda} \sum_{u \in V} a_{vu} x_u$$

其中 $M(v)$ 是节点 v 的相邻节点集, λ 为常数 (特征值). (a_{vu}) 为邻接矩阵. Google 非常有名的网页排序算法 PageRank[19] 就是特征向量中心度的一种变种.

图16列出了各节点的四种中心度. 关于图16节点中心度的计算, 在本章中的第3节 (第10页) 中将给出相关程序.



	C_D	C_C	C_B	C_E
Andre	0.444	0.529	0.023	0.125
Betty	0.444	0.529	0.023	0.125
Carol	0.333	0.500	0.000	0.102
Dave	0.667	0.600	0.102	0.171
Ed	0.333	0.500	0.000	0.102
Fanny	0.556	0.643	0.231	0.142
Garth	0.556	0.643	0.231	0.142
Hale	0.333	0.600	0.389	0.070
Ike	0.222	0.429	0.222	0.017
Jane	0.111	0.310	0.000	0.004

图 16: 社交网络及其标准化中心度, 图片的原型来自 [20]

常用算法及程序实现

在本小节中, 将介绍常用的图论算法及其 Matlab 程序实现. 实际上对于常用的图论算法和网络分析算法, Matlab 有相应的函数可以调用. 在数学建模比赛中, 时间是非常宝贵的, 因此熟练掌握已有函数的调用, 将省去具体算法的实现, 将更多的时间和精力放在解决问题本身上. 在具体介绍具体算法前, 我们先介绍一下图论工具箱和网络分析工具箱. 表2列出了图论工具箱中的函数.

表 2: 图论工具箱的相关命令 [21, 22]

函数名	功能
graphallshortestpaths	求图中所有顶点对之间的最短距离
graphconnredcomp	找无 (有) 向图的 (强/弱) 连通分支
graphisreddag	测试有向图是否含有圈
graphisomorphism	确定一个图是否有生成树
graphmaxflow	计算有向图的最大流
graphminspantree	在图中找最小生成树
graphpred2path	把前驱顶点序列变成路径的顶点序列
graphshortestpath	求指定一对顶点间的最短距离和路径
graphtopoorder	执行有向无圈图的拓扑排序
graphtraverse	求从一顶点出发, 所能遍历图中的顶点

在2.4小节中, 我们已经介绍了使用邻接矩阵来表示无向图和有向图或网络. 邻接矩阵表示法非常简单, 直接. 但是, 通常在邻接矩阵的所有元素中, 只有少量非零元. 当图或网络的规模很大时, 这种情况更加明显. 如果网络比较稀疏, 邻接矩阵表示法浪费大量的存储空间, 从而增加了在网络中查找弧的时间. 为避免这种浪费, 表2列出的图论工具箱的相关命令所使用的数据结构

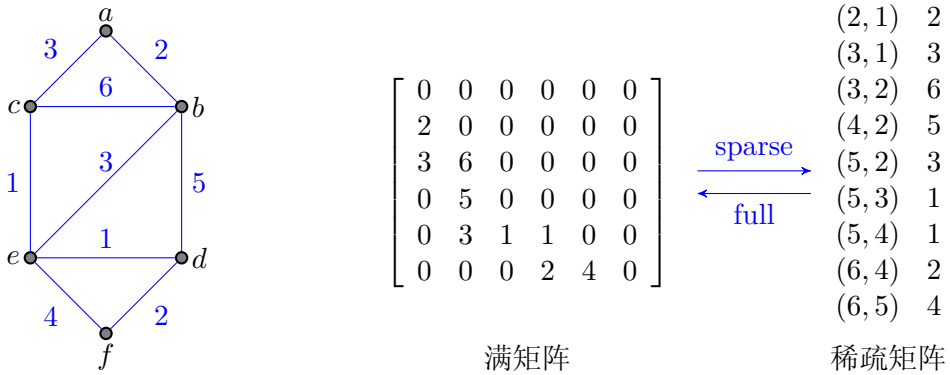


图 17: 赋权无向图, 满矩阵和稀疏矩阵的转化

是稀疏矩阵, 而不是直接使用邻接矩阵. 稀疏矩阵在数学上是指零元素很多, 非零元素很少的矩阵. 对于稀疏矩阵, 只要存放非零元素的位置和值即可, 可以按如下方式存储:

(非零元素的行地址, 非零元素的列地址), 非零元素的值

在 Matlab 中无向图和有向图邻接矩阵的使用是有些差异的:

- 对于有向图, 只要写出邻接矩阵, 直接使用 Matlab 的sparse命令, 就可以把邻接矩阵转化为稀疏矩阵的表示方式.
- 对于无向图, 由于邻接矩阵是对称的, Matlab 中只需使用邻接矩阵的下三角元素, 即 Matlab 只存储邻接矩阵的下三角元素中的非零元素.

稀疏矩阵只是一种存储格式. Matlab 中, 普通矩阵使用 sparse 命令转化为稀疏矩阵, 稀疏矩阵使用 full 命令转化为满矩阵. 图17给出了一个赋权无向图及相应的满矩阵和稀疏矩阵. 下面以图17为例, 给出应用图论工具箱中graphshortestpath函数求解顶点 a(第 1 个顶点) 到顶点 f(第 6 个顶点) 的最短路径[3]的 Matlab 程序 (见代码1).

代码 1: 图论工具箱函数应用实例

```
%     a b c d e f
w = [0 0 0 0 0 0     % a
     2 0 0 0 0 0     % b
     3 6 0 0 0 0     % c
     0 5 0 0 0 0     % d
     0 3 1 1 0 0     % e
     0 0 0 2 4 0];   % f
W = sparse(w);       % 将满矩阵转化为稀疏矩阵
                     % 对于无向图,'Directed'属性设为0
[Dist,Path] = graphshortestpath(W, 1, 6,'Directed',0);
```

代码1给出的计算结果为: Dist = 7, Path = [1 3 5 4 6]. 即顶点 a 到顶点 f 的最短路径长度为 7, 最短路径为 $a-c-e-d-f$.

显然对于网络的分析, 图论工具箱中的函数还太过局限. 接下来, 我们再介绍一个来自 MIT 的非官方网络分析工具箱 [23]. 这个网络分析工具箱集成了很多现成的函数, 最新版本可以从github(具体网址见参考文献 [24]) 网站获得. 由于网络分析工具箱中函数众多, 这里不便一一介绍, 仅列出几个与中心度有关的代表性函数及几个网络可视化函数[4]. 值得注意

表 3: 网络分析工具箱的相关命令 [23]

函数名	功能
degrees	求图中所有顶点的度, 入度和出度.
ave_neighbor_deg	求图中所有顶点的相邻顶点平均度.
closeness	求图中所有顶点的接近中心度.
node_betweenness_faster	求图中所有顶点的中间中心度.
edge_betweenness	求图中所有边的中间中心度.
eigencentrality	求图中所有顶点的特征向量中心度.
clust_coeff	求图中所有顶点的集聚系数.
draw_circ_graph	根据邻接矩阵画出图, 所有节点按度在排在圆周上.
radial_plot	根据给定中心节点, 画辐射图.

的是, 网络分析工具箱所使用的数据结构是满矩阵. 下面以图16为例, 给出网络分析工具箱中degrees和closeness函数求解所有节点的点度中心度和接近中心度的 Matlab 程序 (见代码2).

代码 2: 网络分析工具箱函数应用实例

```
n = 10; % 顶点数

% 给Andre, Betty, ..., Jane标号为1, 2, ..., 10.
Andre = 1; Betty = 2; Carol = 3; Dave  = 4; Ed    = 5;
Fanny = 6; Garth = 7; Hale  = 8; Ike   = 9; Jane  =10;

% 根据图构造邻接矩阵.
A = zeros(10);
A(Andre, [Betty, Carol,  Dave, Fanny]) = 1;
A(Betty, [Andre,  Dave,    Ed, Garth]) = 1;
A(Carol, [Andre,  Dave, Fanny]) = 1;
```

[3]短路径问题的图论提法见第11页中问题1

[4]由于版本的更新, 部分函数名可能有所变化, 这里列出的函数来自 [23]

```
A( Dave, [Andre, Betty, Carol,    Ed, Fanny, Garth]) = 1;
A(   Ed, [Betty,  Dave, Garth]) = 1;
A(Fanny, [Andre, Carol,  Dave, Garth,  Hale]) = 1;
A(Garth, [Betty,  Dave,    Ed, Fanny,  Hale]) = 1;
A( Hale, [Fanny, Garth,   Ike]) = 1;
A(  Ike, [ Hale,  Jane]) = 1;
A( Jane, [  Ike]) = 1;

Cd = degrees(A)' /(n-1)    % 计算点度中心度并标准化.
Cc = closeness(A)*(n-1)    % 计算接近中心度并标准化.
```

图16中还给出了中间中心度和特征向量中心度. 对于中间中心度和特征向量中心度, 读者可以参照以上程序, 利用表3中相关函数进行计算.

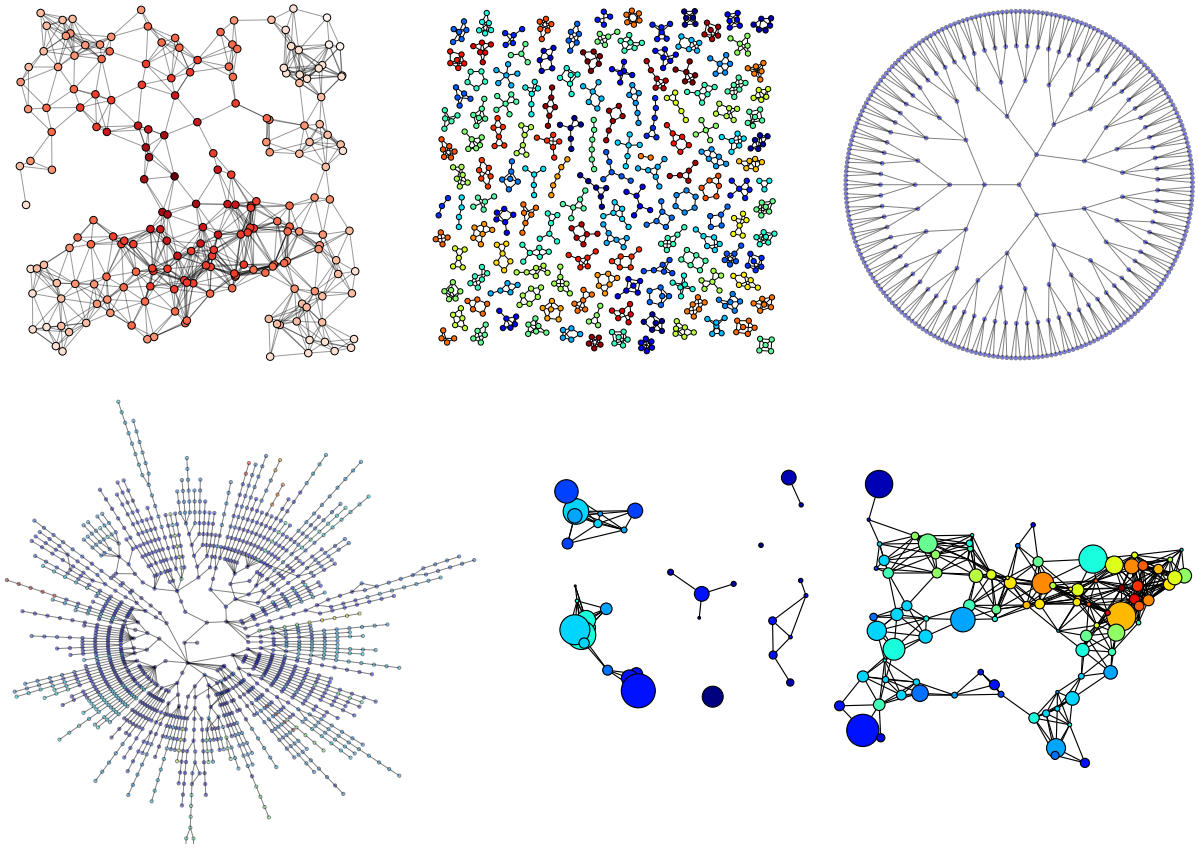


图 18: 由 Python-NetworkX 可视化得到的几种网络图 [25]

以上提到的 Matlab 网络分析工具箱中有关网络可视化函数比较局限, 不能很好的显示出各种网络. 对于有特殊需求的可以利用其它工具来可视化网络, 比如 Python 的 NetworkX, 不仅提供了网络分析的常用函数, 更提供了良好的可视化函数. 图18是由 NetworkX 可视化的几种网络图. 限于篇幅, 这里就不具体介绍 Python 及其 NetworkX 工具包, 想了解更多的读者可以参考网站: `https://networkx.github.io`

最短路径

最短路径问题 [26] 是图论研究中的一个经典算法问题, 旨在寻找图中两顶点之间的最短路径. 数学建模中, 许多优化问题都可以转化为最短路径问题. 最短路径问题的图论提法如下:

问题 1 (最短路径)

对于加权有向图 $G = (V, W)$, 图中各边 (v_i, v_j) 有权 $w(v_i, v_j)$ ($w(v_i, v_j) = \infty$ 表示顶点 v_i 和

v_j 间没有边). 指定两个顶点 v_s 和 v_t, 若存在一条路 μ, 使得它是从 v_s 到 v_t 总权

$$w(\mu) = \sum_{(v_i,v_j)\in\mu} w(v_i, v_j)$$

在所有从 v_s 到 v_t 的路中最小, 则此路径 μ 即为从顶点 v_s 到 v_t 的最短路径.

为了形象的描述, 下面我们给出一个加权有向图, 如图19所示. v_1 到 v_6 的最短路径问题就是要找出一条 v_1 到 v_6 的路, 使得路中的所有边的权之和最小, 也称路径距离最短. 例如我们可以从图19中发现, v_1 到 v_6 的最短距离为 $d(v_1, v_6) = 3+1+1+2 = 7$. 如果图19中的边都无向 (或双向), 则问题变成了加权无向图的最短路径问题. 如果图19中所有边的权都为 1, 便可以看成是普通的 (无权) 有向图.

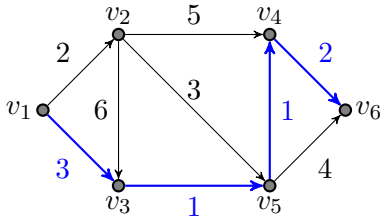


图 19: 加权有向图中的最短路径

如果加权有向图中所有边的权都为正, 则两顶点间的最短路径可以应用 Dijkstra 算法[27]. Dijkstra 算法是目前被认为求无负权网络最短路径的最佳算法. 最短路径具有最优子结构性质: 如果序列 $P(v_i, v_j) = \{v_i, \cdots, v_k, \cdots, v_s, \cdots, v_j\}$ 是从 v_i 到 v_j 的最短路径, 则 $P(v_k, v_s) = \{v_k, \cdots, v_s\}$ 必定是从 v_k 到 v_s 的最短路径. Dijkstra 算法正是基于这种性质, 其基本原理为: 若存在一条从 i 到 j 的最短路径 $\{v_i, \cdots, v_k, v_j\}$, 则序列 $\{v_i, \cdots, v_k\}$ 必为从 v_i 到 v_k 的最短路径. 因此, Dijkstra 算法本质是一种动态规划算法. 为了求出最短路径, Dijkstra 算法以最短路径长度递增, 逐次生成最短路径. 比如对于源顶点 v_1, 首先选择其直接相邻的顶点中长度最短的顶点 v_i, 那么当前已知的从 v_1 到达 v_j 顶点的最短距离为 $d(v_1, v_j) = \min\{d(v_1, v_j), d(v_1, v_i) + w(v_i, v_j)\}$. Dijkstra 算法的具体描述见算法1.

算法 1: Dijkstra

对于加权有向图 $G = (V, W)$, $w(i, j)$ 表示顶点 v_i 和 v_j 间的边权. 源顶点为 v_t, 初始化 $S = \varnothing$ 用于记录已经扫描过的顶点, $d(i)$ 记录 v_t 到 v_i 的最短距离, parent(i) 记录从 v_t 到 v_i 路径上的 v_i 前一个顶点. 则 Dijkstra 算法可表述为:

1. 从差集 $V - S$ 中选择使 $d(i)$ 值最小的顶点 v_i. 将 v_i 加入到 S 中, 即 $S = S \bigcup v_i$.
2. 更新与 v_i 直接相邻顶点的 d 值: $d(j) = \min\{d(j), d(i) + w(i, j)\}$.
3. 如果 $S = V$, 则停止并返回 d 和 parent; 否则跳到1.

下面以图19中所示的加权有向图为例, 给出求解 v_1 到 v_6 最短路径的 Dijkstra 算法的 Matlab 程序, 见代码3. 最终返回的结果为path = [1 3 5 4 6], d = [0 2 3 5 4 7], d(6) = 7. 这表示顶点 v_1 到 v_6 的最短路径为 $v_1 - v_3 - v_5 - v_4 - v_6$, 最短路径的长度为 7.

代码 3: 应用 Dijkstra 算法求解加权图有向图最短路径

```
V = [ 1   2   3   4   5   6];          % 顶点集
w = [ 0   2   3  inf inf inf           % w(i,j) 表示顶点 i 和 j 间的边权
      inf  0   6   5   3  inf
      inf inf  0  inf  1  inf
```

```
      inf inf inf  0  inf  2
      inf inf inf  1   0   4
      inf inf inf inf inf  0];

vs = 1; vt = 6;                          % 最短路径的始末节点
d = inf*ones(1,6); d(vs) = 0;            % 初始化 d 用于记录 v_t 到 v_i 的最短距离
S = [];                                  % 初始化 S=∅ 用于记录已扫描过的顶点
while ¬isempty(setdiff(V,S))             % 如果 V=S, 则停止循环
    U = setdiff(V,S);                    % 差集 U=V-S
    [minU,i] = min(d(U)); vi = U(i);     % 从 U 中先择使 d(i) 值最小的顶点 v_i
    S = [S vi];                          % 将 v_i 加入到 S 中, 即 S=S⋃v_i
    for vj = U
        if d(vi) + w(vi,vj) < d(vj)      % d(j)=min{d(j),d(i)+w(i,j)}
           d(vj) = d(vi) + w(vi,vj);
           parent(vj) = vi;
        end
    end
end

% 通过迭代来回溯出 v_s 到 v_t 的最短路径 path
if parent(vt) ≠ 0
   t = vt; path = vt;
   while t ≠ vs
       p = parent(t);
       path = [p path];
       t = p;
   end
end
```

以上代码中, 函数isempty和 setdiff 是 Matlab 自带函数. isempty用于检验集合 (数组) 是否为空, setdiff 用于求两集合的差集. 实际上我们在了解了算法的本质后, 大何不必自己编写 Dijkstra 算法来解决最短路径问题, 可以应用图论工具箱中的函数直接求解最短路径问题. 甚至对于一些不是很关心算法原理的读者, 完全可以跳过算法的学习, 只要知道函数的输入和输出的数据格式和意义就可以了. 读者可以参考代码1中应用函数graphshortestpath直接求解加权无向图的程序, 来快速的实现加权有向图最短路径的求解.

有些最短路径问题是求网络中任意两节点间最短路径. 这类问题虽然可以用 Dijkstra 算法依次改变起点来计算, 但这样比较繁琐. Floyd 算法 [28] 可以直接求出网络中任意两点间的最短路径. Floyd 算法的本质与 Dijkstra 算法类似, 其原理也是动态规划, 这里不再细述 Floyd 算法, 感兴趣的读者可以参考更多网络资料. 在图论工具箱中, 函数graphallshortestpaths直接求出网络中任意两点间的最短路径长度, 代码4给出了相关程序.

代码 4: 应用图论工具箱中函数graphallshortestpaths求任意两点间的最短路径长度

```
w = [ 0   2   3  inf inf inf          % w(i,j) 表示顶点 i 和 j 间的边权
      inf  0   6   5   3  inf
      inf inf  0  inf  1  inf
      inf inf inf  0  inf  2
      inf inf inf  1   0   4
      inf inf inf inf inf  0];

W = sparse(w);                        % 将满矩阵转化为稀疏矩阵
d = graphallshortestpaths(W);
```

最小生成树

我们在2.3节中已经介绍了树的概念. 本小节将介绍最小生成树 [29] 问题及相关算法. 许多图论和网络的优化问题都可以归结为最小生成树问题, 例如交通系统中设计长度最小的公路网把若干城市联系起来. 最小生成树问题的图论提法如下:

问题 2 (最小生成树)

在一给定的无向图 $G=(V,E)$ 中, (v_i,v_j) 代表连接顶点 v_i 与顶点 v_j 的边 (即 $(v_i,v_j)\in E$), 而 $w(v_i,v_j)$ 代表此边的权重，若存在 T 为 E 的子集 (即 $T\subseteq E$) 且为无循环图, 使得

$$w(T)=\sum_{(v_i,v_j)\in T} w(v_i,v_j)$$

的 $w(T)$ 最小, 则此 T 为 G 的最小生成树. 最小生成树其实是最小权重生成树的简称.

为了形象的描述, 下面我们给出一个加权连通图, 如图20所示. 其最小生成树就是在原图的边子集所构成的树中, 不但包括了连通图里的所有顶点, 且其所有边的权值之和亦为最小. 我们不难发现, 图20的最小生成树为 $T=\{v_1v_6, v_2v_6, v_3v_5, v_4v_5, v_5v_6\}$.

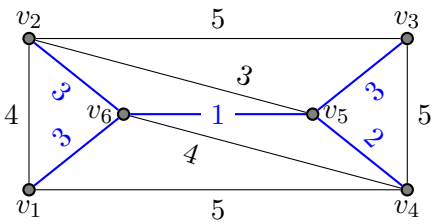


图 20: 加权连通图的最小生成树

对于加权连通图, Prim 算法 [30] 可以有效的搜索其最小生成树. Prim 算法的本质是贪心算法: 从任意一节点开始, 每次挑选的顶点是候选边中权值最小的边的一个端点, 直到所有顶点被挑选完. Prim 算法的具体描述见算法2.

算法 2: Prim

对于一个加权连通图 $G=(V,W)$, 初始化为顶点集 $S=\{x\}$, 边集 $T=\varnothing$, 其中 x 为集合 V 中的任一节点 (起始点). 普里姆算法从单一顶点开始, 按照以下步骤逐步扩大树中所含顶点的数目, 直到遍及连通图的所有顶点.

1. 在集合 W 中选取权值最小的边 (u,s), 其中 s 为集合 S 中的元素，而 u 则是 V 中没有加入 S 的顶点.(如果存在多条具有相同最小权值的边, 可任意选取一条).
2. 将 u 加入集合 S 中, 将 (u,s) 加入集合 T 中.
3. 如果 $S=V$, 则停止并返回集合边集 T; 否则跳至1.

下面以图20中所示的加权连通图为例, 给出求解其最短路径的 Prim 算法的 Matlab 程序, 具体见代码5. 最终返回的结果为T = [1 6; 6 5; 5 4; 6 2; 5 3], 表示最小生成树的边集为 $\{v_1v_6, v_6v_5, v_5v_4, v_6v_2, v_5v_3\}$.

代码 5: 应用 Prim 算法求解加权连通图最小生成树

```
w = [ 0   4   inf   5   inf   3          % w(i,j) 表示顶点 i 和 j 间的边权
      4   0    5   inf   3    3
     inf  5    0    5    3   inf
```

```
       5   inf   5    0    2    4
      inf   3    3    2    0    1
       3    3   inf   4    1    0];

S = [1]; T = [];                          % 初始化为 S = {1}, T = ∅
while ¬isempty(setdiff(V,S))              % 如果 S = V, 则停止
   U = setdiff(V,S);                      % U = V − S, V 中没有加入 S 的顶点

   Wsu = w(S,U);
   [I,J] = find(Wsu==min(Wsu(:)));        % 找出所有权值最小的边
   s = S(I(1)); u = U(J(1));              % 取第一条权值最小的边 (u, s)

   S = [S u];                             % 将 u 加入集合 S 中
   T = [T; u, s];                         % 将 (u, s) 加入集合 T 中
end
```

求解加权连通图最小生成树的经典算法还有 Kruskal 算法 [31]. Kruskal 算法的本质是也是贪心算法: 从剩下的边中选择一条不会产生环路的具有最小权的边加入已选择的边的集合中, 注意到所选取的边若产生环路则不可能形成一棵生成树. 这里不再细述 Kruskal 算法, 感兴趣的读者可以参考更多网络资料. 图论工具箱中提供了graphminspantree函数来求解加权连通图的最小生成树. 函数graphminspantree可以指定求解加权连通图最小生成树的算法为 Prim 或 Kruskal. 代码6给出了应用函数graphminspantree求解图20所示的加权连通图的最小生成树的 Matlab 程序. 需要指出的是graphminspantree给出的最小树ST的表示方法是稀疏矩阵.

代码 6: 应用图论工具箱中函数graphminspantree求解加权连通图最小生成树

```
w = [ 0   4   inf   5   inf   3          % w(i, j) 表示顶点 i 和 j 间的边权
      4   0    5   inf   3    3
     inf  5    0    5    3   inf
      5  inf   5    0    2    4
     inf  3    3    2    0    1
      3   3   inf   4    1    0];

W = sparse(w);                           % 将满矩阵转化为稀疏矩阵
[ST, pred] = graphminspantree(W);
```

最大流问题

本节将介绍最大流问题 [32] 及相关算法. 许多系统包含了流量问题. 例如交通系统有车流量, 金融系统有现金流, 控制系统有信息流等. 在介绍最大流问题前, 首先简单介绍一下网络流的相关概念.

定义网络 $N = N(s,t)$ 由图 $D(V,A)$ 及一个定义在弧 A 上的非负函数 C 构成. 该网络有一个源 s 和一个汇 t. 函数 C 称为容量函数, 函数 C 在弧 (u,v) 上的值 c_{uv} 称为弧 (u,v) 的容量. 如果我们描述物流, 节点 s 相当于物品的供应点, 节点 t 相当于物品的接收点, 其它节点称为中间点, 某边的容量表示该边所能运输物品的最大速率. 物流过程就是物品从供应点 s 经过中间点向接收点 t 的实体流动过程. 图21是一个网络流的实例. 对于网络 N, 其上的一个流 f 是指从 N 的弧集 A 到实数 R 的一个函数, 即对每条弧 (u,v) 赋予一个实数 f_{uv}, 称为弧 (u,v) 的流量. 对于任间中间节点 v 满足

$$f^+(v) - f^-(v) = \sum_{u:(u,v)\in A} f_{uv} - \sum_{u:(v,u)\in A} f_{vu} = 0 \tag{1}$$

其中 $f^+(v)$ 表示从节点 v 流出的流量, $f^-(v)$ 表示流入节点 v 的流量. 式 (1) 要求对于任何中间节点流入的流量之和必须等于从该点流出的流量之和, 即流入的流量之和与流出的流量之和的差为零, 也就是说各中间点只起转运作用, 它既不产出新的物资, 也不得截留过境的物资. 例

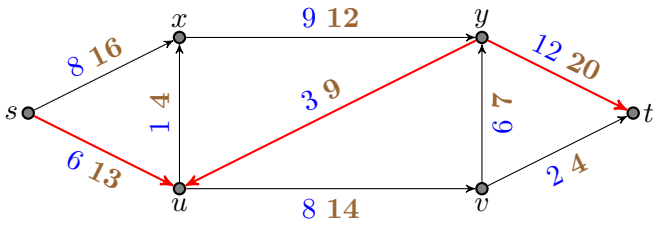


图 21: 一个网络流实例: 流量, 容量和增广路径

如在图21中, 对于中间节点 v 有 $f^+(v)-f^-(v)=(2+6)-8=0$. 对于源 x 和汇 y, 源发出的流的总和 (称为流量), 必须等于汇接收的流的总和

$$f^+(x)-f^-(x)=-\big(f^+(y)-f^-(y)\big)=\mathrm{var}(f) \tag{2}$$

流 f 的流量一般记为 $\mathrm{var}(f)$. 例如在图21中, 对于源 s 有 $f^+(s)-f^-(s)=(6+8)-0=14$, 对于汇 t 有 $f^+(t)-f^-(t)=0-(2+12)=-14$, 因此图21中的流量为 $\mathrm{var}(f)=14$. 对于每一条边, 还要满足容量限制

$$0\le f_{uv}\le c_{uv} \tag{3}$$

以上式 (3) 要求每条弧上的流量必须是非负的且不能超过该弧的最大通过能力 (即该弧的容量). 例如在图21中, 对于弧 (uv), 其上的流量为 8, 容量为 14, 满足流量小于容量 $(8<14)$ 的限制条件. 对于网络 N 上的一个流 $f=\{f_{uv}\}$, 若同时满足式 (1), 式 (2) 和式 (3), 则称 f 为可行流, $\mathrm{var}(f)$ 为可行流量.

对于一个给定的可行流 $f=\{f_{uv}\}$, 将 $r_{uv}=c_{uv}-f_{uv}$ 称为边 (u,v) 的残余容量, 由此构成的网络称为**残留网络**. 把 $f_{uv}=c_{uv}$ 的弧称为饱和弧, $f_{uv}<c_{uv}$ 的弧称为非饱和弧, $f_{uv}=0$ 的弧称为零流弧. 若 μ 是网络中连接源点 s 和汇点 t 的一条路径, 定义路径的方向为从 s 到 t, 则路径上的弧可分为两类: 一类是弧的方向与路径的方向一致, 叫做前向弧, 前向弧的全体记为 μ^+; 另一类弧与路径的方向相反, 称为后向弧, 后向弧的全体记为 μ^-. 若路径 μ 满足: 前向弧是非饱和弧, 后向弧是非零流弧, 则 μ 为可行流 f 的一条**增广路径**. 若令

$$\epsilon=\min\{\epsilon_{uv}:(u,v)\in\mu\},\qquad \text{其中}\epsilon_{uv}=\begin{cases} r_{uv}=c_{uv}-f_{uv}, & (u,v)\in\mu^+\\ f_{uv}, & (u,v)\in\mu^-\end{cases} \tag{4}$$

则在增广路径 μ 上的每条前向弧的流都可以增加一个量 ϵ, 而相应的后向弧的流可减少 ϵ, 这样就可使得网络的流量增加 ϵ. 同时可以使每条弧的流量不超过它的容量, 而且保持为正, 也不影响其它弧的流量. 例如图21中, 路径 $\{(s,u),(u,y),(y,t)\}$ 就是一条增广路径, 其中 (s,u) 和 (y,t) 为前向弧, (u,y) 为后向弧. 通过该条增广路径, 网络的流量可增加 3.

所谓最大流问题, 就是在给定网络的条件下, 寻找最大的可行流. 许多流问题都是要确定系统网络所能承受的最大流量以及如何达到这个最大流量. 最大流问题的图论提法如下:

问题 3 (最大流问题)

对于给定的网络 $N=N(s,t)$, 求从 s 到 t 的最大可行流. 其中网络 N 由图 $D(V,A)$ 及一个定义在弧 A 上的容量函数 C 构成. 用线性规划的方法, 最大流问题可以描述为

$$\begin{aligned}\max\quad & \mathrm{var}(f)\\ \text{s.t.}\quad & \sum_{v:(v,u)\in A}f_{vu}-\sum_{v:(u,v)\in A}f_{uv}=\begin{cases}\mathrm{var}(f) & v=s\\ -\mathrm{var}(f) & v=t\\ 0 & v\ne s,t\end{cases}\\ & 0\le f_{uv}\le c_{uv},\quad \forall(u,v)\in A\end{aligned}$$

计算最大流最基本的方法是 Ford-Fulkerson 方法 [33] 和 Edmonds-Karp 算法 [34]. 由于不同的具体实现方案, 其执行效率不同. Ford-Fulkerson 方法的思想是: 先找到一条从源点到汇点的增广路径, 这条路径的容量是其中容量最小的边的容量. 然后, 通过不断找增广路, 一步步扩大流量, 当找不到增广路时, 就得到最大流了. Ford-Fulkerson 方法有一定的局限性, 在实际中并不常用. 当边的容量是实数而非整数时, Ford-Fulkerson 方法将永远执行不完. Edmonds-Karp 算法通过每次寻找残留网络中最短路径的方法对 Ford-Fulkerson 算法进行了改进. Edmonds-Karp 算法的具体描述见算法3.

算法 3: Edmonds-Karp

对于一个由有向图 $D(V,A)$ 及定义在其弧 A 上的容量函数 C 构成的网络 $N=N(s,t)$, 初始化各边流量 $f_{uv}=0, \forall(u,v)\in A$, 初始化最大流 $f_{\max}=0$. 执行以下循环直到残留网络 $r_{uv}=c_{uv}-f_{uv}$ 中再也找不到从 s 到 t 的路径:

1. 由广度优先搜索 (BFS) 找出残留网络 r_{uv} 中从 s 到 t 的一条最短路径 μ.
2. 以 μ 为增广路径增加流量 $f_{uv}=f_{uv}+\epsilon$ 和 $f_{\max}=f_{\max}+\epsilon$.
3. 更新残留网络 $r_{uv}=c_{uv}-f_{uv}$.

返回各边流量 f_{uv} 和最大流 $f_{\max}$.

由于 Edmonds-Karp 算法涉及广度优先搜索 [35] 算法, 这里不给出实现 Edmonds-Karp 算法的 Matlab 程序. 实现 Edmonds-Karp 算法的完整 Matlab 代码可以在 Wiki 网站 [36] 上找到. 图论工具箱中提供了graphmaxflow函数来求解网络中的最大流问题. 函数graphmaxflow可以指定求解网络中的最大流的算法为 Edmonds(即 Edmonds-Karp) 或 Goldberg. 代码7给出了应用函数graphmaxflow求解图21所示的网络中最大流的 Matlab 程序.

代码 7: 应用图论工具箱中函数graphmaxflow求解网络中的最大流

```
    % s  x  y  u  v  t                    % 容量矩阵 C = (c_uv)
C = [ 0 16  0 13  0  0   % s
      0  0 12  0  0  0   % x
      0  0  0  9  0 20   % y
      0  4  0  0 14  0   % u
      0  0  7  0  0  4   % v
      0  0  0  0  0  0]; % t

c = sparse(C);                           % 将满矩阵转化为稀疏矩阵
[fmax,flow] = graphmaxflow(c, 1, 6, 'method', 'Edmonds')
```

问题3中对于最大流问题的描述, 采用了线性规划的方法. 实际上, 很多图论问题都可以用线性规划的方法描述, 也自然能够用线性规划来求解. 在 Matlab 中可用linprog函数来求解线性规划问题. 因此, 图21所示的网络中的最大流也可采用以下代码来求解:

代码 8: 应用线性规划求解网络中的最大流

```
    % sx su xy yu yt ux uv vy vt      % 关联矩阵
M = [-1 -1  0  0  0  0  0  0  0      % s
      1  0 -1  0  0  1  0  0  0      % x
      0  0  1 -1 -1  0  0  1  0      % y
      0  1  0  1  0 -1 -1  0  0      % u
      0  0  0  0  0  0  1 -1 -1      % v
      0  0  0  0  1  0  0  0  1]     % t

                                     % 目标函数为源点出流量最大
C = [-1 -1  0  0  0  0  0  0  0];    %     min  C * flow

```

```
A = [];                                  % 不等式约束: A * flow < b
b = [];                                  % 没有等式约束, 所以A, B 都为[]

Aeq = M(2:5,:);                          % 等式约束: Aeq * flow = beq
beq = zeros(4,1);

lb= [ 0  0  0  0  0  0  0  0  0]';       % 下界: flow ≥ lb
ub= [16 13 12  9 20  4 14  7  4]';       % 上界: flow ≤ ub

flow = linprog(f, A, b, Aeq, beq, lb, ub);
fmax = -f'*flow
```

需要指出的是, Lingo 或 Lindo 这类专门求解线性和通用优化问题的语言可以更为方便地求解线性规划问题. 在司守奎等人所编的 “数学建模算法与应用”[22] 一书中给出了大量用 Lingo 语言求解图论的例子. 由于本书编程语言的统一性, 这里不再给出具体的 Lingo 或 Lindo 的相关代码.

最小费用流

在3.3节中, 我们介绍了求解网络中两节点间最大可行流的算法, 但是还没有考虑到网络中可行流的费用问题, 在许多实际问题中, 费用的因素很重要. 例如, 在运输问题中, 人们总是希望在完成运输任务的同时, 寻求一个使总的运输费用最小的运输方案. 这就是下面要介绍的最小费用流问题 [37]. 所谓最小费用流问题, 就是在指定网络流量的情况下, 寻求最小费用的可行流. 最小费用流问题的图论提法如下:

问题 4 (最小费用流)

对于给定的网络 $N = N(s,t)$, s 和 t 分别为源点和汇点, 该网络由有向图 $D(V,A)$ 及定义在弧 A 上的容量函数 C 和费用函数 W 构成. w_{uv} 即为弧 (u,v) 上单位流量的费用. 最小费用流问题就是寻求从 x 到 y 流量为 d 的一个可行流, 使得该可行流费用最小. 用线性规划的方法, 最小费用流问题可以描述为

$$\begin{aligned}
\min \quad & \sum_{(u,v)\in A} w_{uv} f_{uv} \\
\text{s.t.} \quad & \sum_{v:(v,u)\in A} f_{vu} - \sum_{v:(u,v)\in A} f_{uv} = \begin{cases} d & v = s \\ -d & v = t \\ 0 & v \neq s,t \end{cases} \\
& 0 \le f_{uv} \le c_{uv}, \quad \forall (u,v) \in A
\end{aligned}$$

如果指定的网络流量 d 恰好是网络的最大流 $\mathrm{var}(f_{\max})$, 则问题就是所谓的最小费用最大流问题. 若 $d > \mathrm{var}(f_{\max})$, 则该问题无解. 与最大流问题一样, 最小费用流问题中, 目标函数是线性的, 约束条件也是线性, 因此同样可以应用线性规划来求解最小费用流问题.

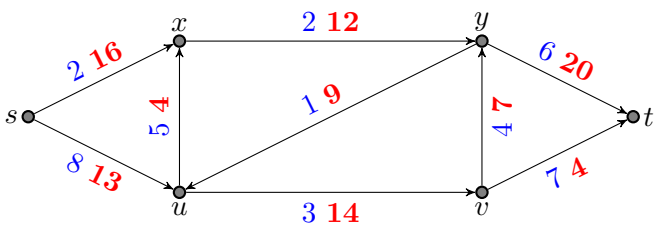


图 22: 最小费用流: 费用, 容量

这里我们以图22所示的网络图为例, 应用线性规划求解从 s 到 t 的流量为 23 的最小费用流. 代码9给出了调用函数linprog求解该最小费用流问题的 Matlab 代码.

代码 9: 应用线性规划求解网络中的最小费用流

```
    % sx su xy yu yt ux uv vy vt      % 关联矩阵
M = [-1 -1  0  0  0  0  0  0  0      % s
      1  0 -1  0  0  1  0  0  0      % x
      0  0  1 -1 -1  0  0  1  0      % y
      0  1  0  1  0 -1 -1  0  0      % u
      0  0  0  0  0  0  1 -1 -1      % v
      0  0  0  0  1  0  0  0  1]     % t

                                     % 目标函数为总费用最小
C = [ 2  8  2  1  6  5  3  4  7];    %     min C * flow

f = 23;                              % 指定网络流量

A = [];                              % 不等式约束: A * flow < b
b = [];                              % 没有等式约束, 所以A, B 都为[]

Aeq = M;                             % 等式约束: Aeq * flow = beq
beq = [-f  0  0  0  0  f]';

lb= [ 0  0  0  0  0  0  0  0  0]';   % 下界: flow ≥ lb
ub= [16 13 12  9 20  4 14  7  4]';   % 上界: flow ≤ ub

flow = linprog(C, A, b, Aeq, beq, lb, ub)
cost = C*flow
```

图论工具箱中没有提供直接求解最小费用流问题的函数. 最小费用流问题还可以应用 Busacker-Gowen 等算法 [38, 37] 来求解, 但较之于线性规划, 无论是算法的描述还是程序的实现上都烦杂得多, 这里不再给出其它算法和代码, 感兴趣的读者可以自行了解其它相关算法.

最小 Hamilton 回路问题 (TSP 问题)

本节将介绍最小 Hamilton 回路问题 [39]. 所谓 Hamilton 回路就是从图的一个顶点出发, 途中经过所有其他节点且只经过一次并最终回到出发点的路径. 而最小 Hamilton 回路问题则是寻找图的所有 Hamilton 回路中最短的一个. 数学建模比赛中经常直接或间接涉及的旅行商 (TSP) 问题 [40] 就是最小 Hamilton 回路问题, 旅行商问题是图论中著名经典问题之一, 假设有一个旅行商人要选择一条路径拜访 n 个城市, 限制是每个城市只能拜访一次, 最后要回到原来出发的城市, 目标是要求所选路径路程为所有路径之中的最小值. 最小 Hamilton 回路的图论提法如下:

问题 5 (最小 *Hamilton* 回路问题)

对于给定的加权无向图 $G=(V,W)$, 寻找 G 中的 *Hamilton* 回路 C, 使得 C 的总权

$$w(C)=\sum_{(v_i,v_j)\in C} w(v_i,v_j)$$

最小. 若用 x_{ij} 表示经过各节点间的路径 *(路径是有向的)*:

$$x_{ij}=\begin{cases}0 & (v_i,v_j)\notin C\\ 1 & (v_i,v_j)\in C\end{cases}$$

则最小 *Hamilton* 回路问题可用线性规划的方法描述为

$$\min \quad \sum_{i\neq j}^{n} w_{ij}x_{ij} \tag{5}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij}=1, \quad i=1,\cdots,n \tag{6}$$

$$\sum_{i=1}^{n} x_{ij}=1, \quad j=1,\cdots,n \tag{7}$$

$$x_{ij}\in\{0,1\} \tag{8}$$

$$\sum_{i,j\in S} x_{ij}\leq |S|-1, \quad S\subsetneq V,\ |S|>1 \tag{9}$$

式 *(6)* 约束条件表示每个节点只有一条出弧, 式 *(7)* 约束条件表示每个节点只有一条入弧. 式 *(8)* 约束条件表示 x_{ij} 为 *0* 或 *1* 的整数变量. 式 *(9)* 约束条件表示任何顶点集 V 的真子集都不构成回路.

问题5中对最小 Hamilton 回路问题的描述, 采用了整数规划的方法, 但式 (9) 约束条件并不太容易表示, 在用计算机实现整数规划时通常将式 (9) 替换为 Miller-Tucker-Zemlin (MTZ) 约束条件 [41]:

$$u_i-u_j+nx_{ij}\leq n-1, \quad 1<i\neq j\leq n \tag{10}$$

其中 $n=|V|$ 为顶点的数量. 式 (10) 的约束也可以保证各边不构成回路: 若 i, j 两点构成回路, 则 $x_{ij}=1$, $x_{ji}=1$, 根据式 (10) 有 $u_i-u_j\leq -1$ 且 $u_j-u_i\leq -1$, 从而 $0\leq -2$, 导致矛盾; 若 i, j, k 三点构成回路, 则 $x_{ij}=1$, $x_{jk}=1$, $x_{ki}=1$, 根据式 (10) 有 $u_i-u_j\leq -1$, $u_j-u_k\leq -1$, $u_k-u_i\leq -1$, 从而 $0\leq -3$, 导致矛盾; 更多节点构成回路的情况以此类推. 因此, 将式 (9) 替换为式 (10) 就可以利用 Matlab 中整数规划函数 intlinprog 来直接求解, 也有不少国内的数模教材 [22, 42] 给出了这种方式的 Lingo 解法.

以上求解方法需要的变量太多, 对于节点数较多的情况, 需要较长的计算时间. 这里给出一种改进的整数规划方式, 见算法4.

算法 4: 最小 Hamilton 回路的改进整数规划

对于无向图, 可不考虑路径的方向, 即不必区分 x_{ij} 和 x_{ji}, 只考虑其中一个变量即可, 不妨只考虑 $i<j$ 的情况. 只要 i 和 j 间存在边, 则 $x_{ij}=1$ $(i<j)$. 因此式 (5-8) 可改写为

$$\min \quad \sum_{i<j}^{n} w_{ij}x_{ij} \tag{11}$$

$$\text{s.t.} \quad \sum_{j<k} x_{ik} + \sum_{j>k} x_{kj} = 2, \quad k=1,\cdots,n \tag{12}$$

$$x_{ij} \in \{0,1\}, \quad i<j \tag{13}$$

改写后的变量 $\mathbf{X}=\{x_{ij}\}$ 中的元素减少了一半. 但式 (11-13) 没有防止构成回路的约束. 假设采用整数规划的方式求解得到 m 个子圈的顶点集 $S_1, S_2, \cdots S_m \subset V$, 则依次执行以下操作直到只含一个圈, 即 $m=1$.

- 增加 m 个不等式约束

$$\sum_{i,j\in S_k,\ i<j} x_{ij} \le |S_k|-1, \quad k=1,\cdots,m \tag{14}$$

  重新用整数规划求解. 需要注意的是, 这个约束与式 (9) 的形式和原理都非常相似.

- 根据整数规划重新求解出的结果, 确定解包含的子圈个数 m, 及 m 个子圈的顶点集 $S_1, S_2, \cdots S_m \subset V$.

根据算法4, 并参考 [43] 中的程序, 我们可以自定义一个函数minhamiltonpath用来求解最小 Hamilton 回路问题, 具体见代码10.

代码 10: 求解最小 Hamilton 回路的自定义函数 minhamiltonpath

```
function [order, totdist] = minhamiltonpath(D)
N = size(D, 1);                          % 节点数
idxs = nchoosek(1:N, 2);                 % 边集: 所有 1:N 中的两数组合 (i,j), i<j
lendist = length(dist);                  % 边数, 也是向量 x 的长度

%% 目标函数式 (11): 转化为向量形式 min dist × x
dist = D(sub2ind([N, N], idxs(:, 1), idxs(:, 2)));

%% 等式约束式 (12): 转化为向量形式 Aeq × x = beq
Aeq = spalloc(N, length(idxs), N*(N-1));
for i = 1:N
    whichIdxs = idxs(:,1) == i | idxs(:,2) == i;
    Aeq(i,whichIdxs) = 1;
end
beq = 2*ones(N, 1);

%% 整数约束式 (13): 转化为向量形式 lb ≤ x ≤ ub, 且 x 取整
intcon = 1:lendist;                      % 向量 x 中 1:lendist 个元素为整数
lb = zeros(lendist, 1);                  % 下界
ub = ones(lendist, 1);                   % 上界

opts = optimoptions('intlinprog','Display','off'); % 执行规划时不输出信息

%% 求解整数规划
x = intlinprog(dist,intcon,[],[],Aeq,beq,lb,ub,opts);
```

```
26
27 tours = detectSubtours(x, idxs);      % 各子圈顶点集 tours{1}, tours{2} ...
28 ntours = length(tours);               % 子圈数
29
30 %% 不等约束式 (14): 添加不等约束, 直到只含一个圈 A × x ≤ b
31 A = spalloc(0, lendist, 0);           % 初始为 A 为空
32 b = [];                               % 初始为 b 为空
33 while ntours > 1                       % 只要子圈数大于 1 则执行循环
34     fprintf(1, '%i subtours\n', ntours);          % 在命令窗口中输出子圈数
35     for i = 1:length(tours)
36         rowIdx = size(A, 1)+1;
37         touri = tours{i};             % 第 i 个子圈的顶点集
38         % variations 为 touri 中 2 元素索引组合, 即 touri 中可形成边的两顶点索引组合
39         variations = nchoosek(1:length(touri), 2);
40         for j = 1:length(variations)
41             % 找出边 touri(variations(j,:)) 在边集 idxs 中的位置 whichVar
42             whichVar = (sum(idxs==touri(variations(j,1)),2)) & ...
43                        (sum(idxs==touri(variations(j,2)),2));
44             A(rowIdx, whichVar) = 1; % 添加一行, 使 whichVar 位置为 1
45         end
46         b(rowIdx) = length(touri)-1; % 限制子圈顶点间边数比顶点数小 1
47     end
48     x = intlinprog(dist,intcon,A,b,Aeq,beq,lb,ub,opts); % 重新求解整数规划
49     tours = detectSubtours(x, idxs); % 各子圈顶点集 tours{1}, tours{2} ...
50     ntours = length(tours);          % 子圈数
51 end
52
53 totdist = dist' * x;                 % 求总权 (路程)
54 order = tours{:};                    % 给出相应的路径节点顺序
```

我们以中国 34 个省会城市为例, 并利用代码10定义的minhamiltonpath函数来求解最小 Hamilton 回路 (TSP) 问题. 具体 MatLab 求解程序见代码11, 计算结果见图23.

代码 11: 应用线性规划求解网络中的最小费用流

```
1  lat = [ 39.54  31.14  39.09  29.32  45.45 ...      % 省会城市的纬度
2          43.52  41.50  40.49  38.02  37.52 ...
3          36.38  34.48  34.16  36.03  38.20 ...
4          36.38  43.48  31.51  32.02  30.14 ...
5          28.11  28.41  30.37  30.39  26.35 ...
6          26.05  23.08  20.02  22.48  25.00 ...
7          29.39  22.18  22.14  25.03         ];
8
9  lon = [116.28 121.29 117.11 106.32 126.41 ...      % 省会城市的经度
10         125.19 123.24 111.48 114.28 112.34 ...
11         117.00 113.42 108.54 103.49 106.16 ...
12         101.45  87.36 117.18 118.50 120.09 ...
13         113.00 115.52 114.21 104.05 106.42 ...
14         119.18 113.15 110.20 108.20 102.41 ...
15          90.08 114.10 113.35 121.31         ];
16
17 n = length(lat);                          % 城市 (节点) 数量
18
19 %% 求距离矩阵 dist: dist(i,j) = i 城市与 j 城市 (i < j) 间的球面距离
20 R = 6378.137;                             % 地球半径
21 dist = zeros(n);                          % 初始化距离矩阵
22 for i = 1:n
23     for j = i+1:n
```

```
        dist(i,j) = distance(lat(i),lon(i), lat(j),lon(j), R);
    end
end

%% 调用自定义函数 minhamiltonpath 直接求解最小 hamilton 回路
[order,totdist] = minhamiltonpath(dist)
plot(lon(order([1:end,1])), lat(order([1:end,1])),'o-') % 画路径
```

对于这样一个节点规模为 34 的最小 Hamilton 回路 (TSP) 问题, MatLab 计算时间约为 0.5 秒. 经测试, 对于节点规模为 200 的算例, Matlab 的计算时间也仅为 1 分钟左右; 当节点规模增加

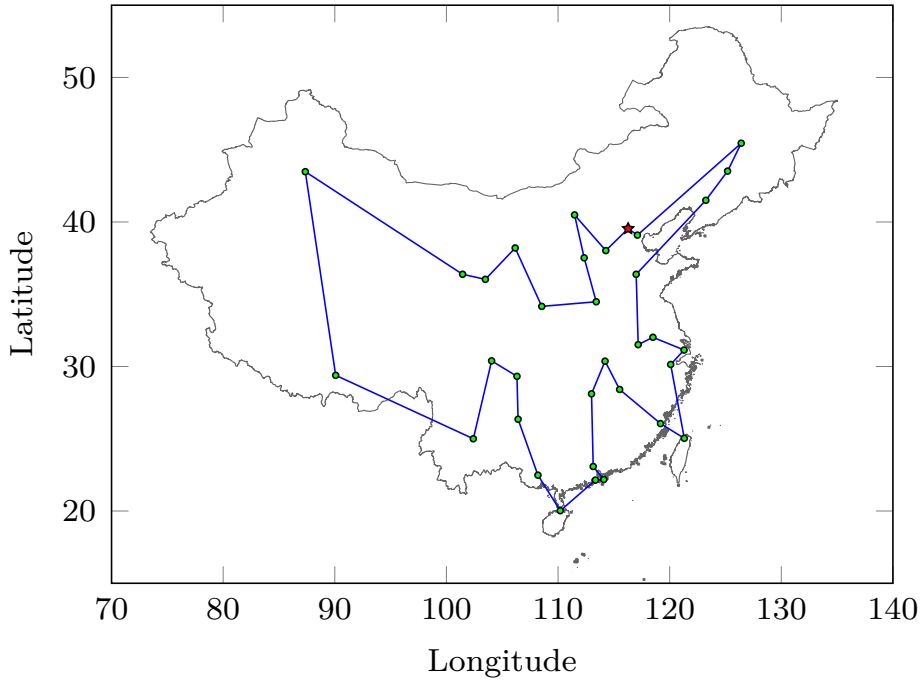


图 23: 中国省会城市的最小 Hamilton 回路 (TSP) 问题的解: 总路程 15651.84 km

至 300 时, MatLab 的计算时间上升至 1 小时左右; 对于节点规模更大的情况, 读者可以改用运行速度更快的编译型语言 (如 C++), 并参考更为高效的算法 (如 Dantzig-Fulkerson-Johnson 算法 [44]). 对于节点规模极大的情况, 可以利用启发式算法 (如退火算法 [45]) 在可接受的计算费用内去寻找尽可能好的解.

应用举例

在数学建模竞赛及现实生活中, 很多问题都可以抽象为由节点和边构成的图或网络模型, 相关函数可以定义在节点或边上. 这类问题的求解通常转化为求解图或网络中满足某些属性的边和节点, 或边和节点的某些属性. 在本节中, 我们分别列举一个图论与网络模型在数学建模竞赛中的应用实例 “灾情巡视路经问题” 和商业中的应用实例 “网页排序算法 PageRank”.

灾情巡视路径问题 (CUMCM1998B)

灾情巡视路径问题 [46] 是 1998 年全国大学生数学建模竞赛的 B 题, 原题如下: 图24为某县的乡 (镇), 村公路网示意图, 公路边的数字为该路段的公里数. 1998 夏天该县遭受水灾. 为考察灾情, 组织自救, 县领导决定, 带领有关部门负责人到全县各乡 (镇), 村巡视. 巡视路线指从县政府所在地出发, 走遍各乡 (镇), 村, 又回到县政府所在地的路线.

- 若分三组 (路) 巡视, 试设计总路程最短且各组尽可能均衡的巡视路线.
- 假定巡视人员在各乡 (镇) 停留时间 $T = 2$ 小时, 在各村停留时间 $t = 1$ 小时, 汽车行驶速度 $v = 35$ 公里/小时. 要在 24 小时内完成巡视, 至少应分几组; 给出这种分组下你认为最佳的巡视路线.
- 在上述关于 T, t 和 v 的假定下, 如果巡视人员足够多, 完成巡视的最短时间是多少; 给出在这种最短时间完成巡视的要求下, 你认为最佳的巡视路线.
- 若巡视组数已定 (如三组), 要求尽快完成巡视, 讨论 T, t 和 v 改变对最佳巡视路线的影响.

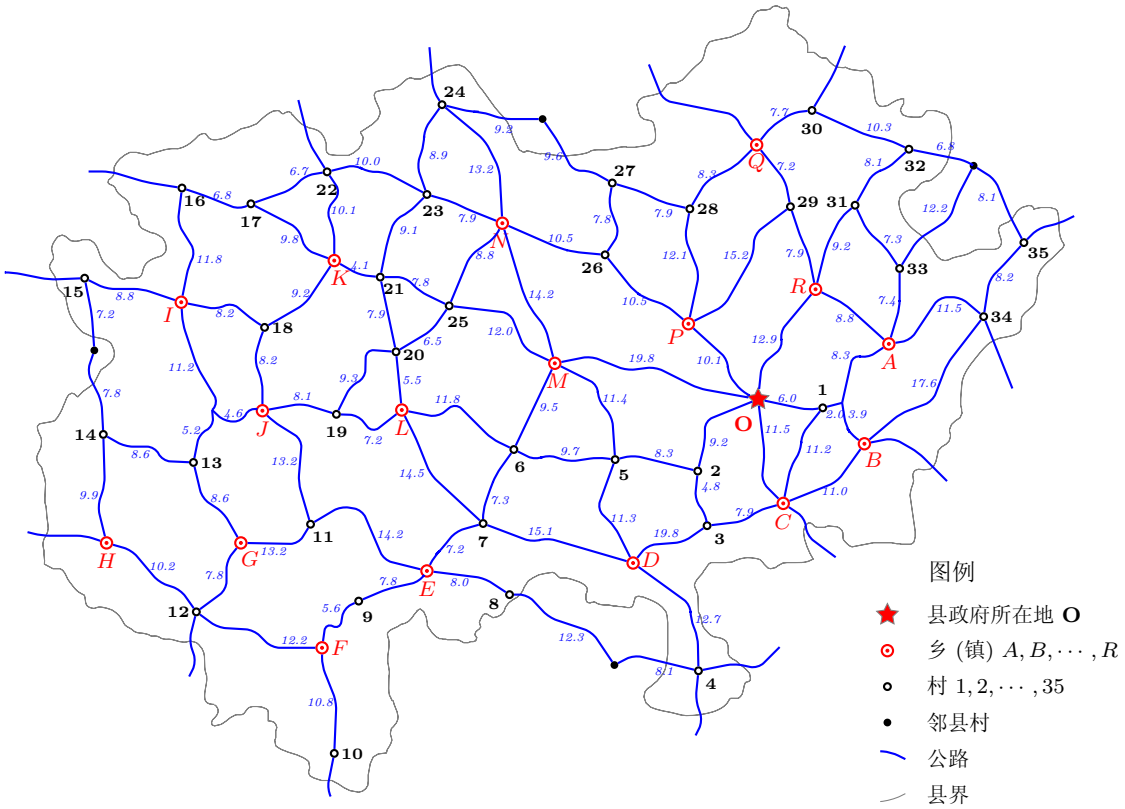


图 24: 灾情巡视路经问题 (CUMCM1998B)

该题是一道典型的图论问题, 其求解过程涉及多种图论问题, 如最短路径, 最小生成树, 最小 Hamilton 回路等问题. 本节仅详细讨论并求解该题的前两问, 讨论和求解过程参考了一篇优秀参赛论文 [47] 和出题人给出的评论 [48].

本题给出了某县的公路网络图, 要求的是在不同的条件下, 灾情巡视的最佳分组方案和路线. 将每个乡 (镇) 或村看作一个图的顶点, 各乡镇, 村之间的公路看作此图对应顶点间的边, 各条公路的长度 (或行驶时间) 看作对应边上的权, 所给公路网就转化为加权网络图. 我们根据题目中所给的公路图, 编写函数 road2graph 用来生成距离矩阵, 具体程序见代码12, 相应的加权网络图见图25. 通过代码12将题目中的公路图转化为加权图, 问题就转化为一类图上的点的行遍性问题, 也就是要用若干条环路覆盖图上所有的顶点, 并使某些指标达到最优.

点的行遍性问题在图论中称为 Hamilton 回路问题, 也称为旅行商 (TSP) 问题. 关于 Hamilton 回路或旅行商 (TSP) 问题的定义, 参看3.5小节. 本题要求的是分组巡视的最佳路径, 与多旅行商问题 (MTSP) 或车辆路径问题 (VRP)[49] 类似, 即要求 m 条经过同一点 O 并覆盖所有其他顶点, 又使边权之和达到最小的环路. 如第一问中 m 取值为 3. 求解非完全图的多旅行商问题, 通常所用的方法可分为两步:

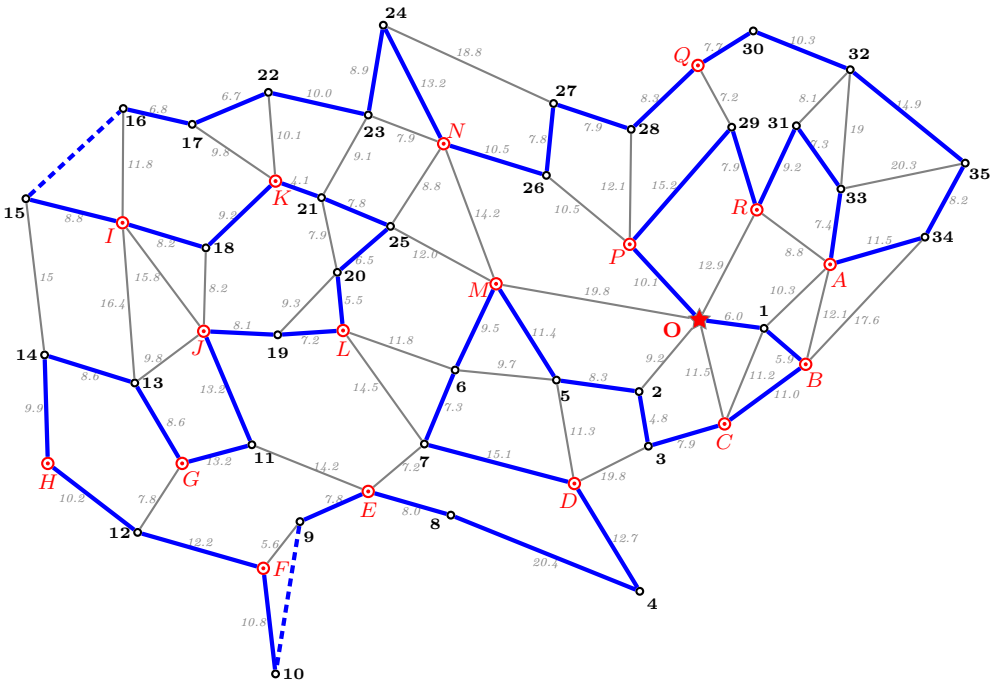


图 25: 交通图对应的加权网络图 $G(V,E)$ 及单旅行商的最优路线 (508.6km)

1. 首先是利用任意两点间的最短路径长度作为该两点间的权构造一个完全图. 这一点对于原图中没有边相连的点对尤为重要. 容易证明, 如此构造出来的完全图, 各边权满足三角不等式, 即任意三点间, 两边权之和不小于第三边的权; 并且该完全图中的最短 Hamilton 回路问题与原图等价.

2. 问题转化为在完全图中求解以 O 为始末点的多旅行商问题, 即在完全图中, 求解 m 条经过同一点 O 并覆盖所有其他顶点, 又使边权之和达到最小的回路.

值得注意的是多旅行商问题的最优解是使 m 条旅行路线的总路程达到最小, 而这 m 条路线的均衡性可能相当差. 本题要求均衡性较好的解, 则还需要做一定的调整工作. 对于单旅行商问题, 我们在3.5节中已经给出了求解方法, 可直接调用代码10定义的minhamiltonpath函数, 代码13是图25的单旅行商问题求解程序, 程序求解得到的最优路径已经在图25中标出, 其中虚线表示两点间没有直接相连的公路, 需要经过中间点到达. 容易证明, 单旅行商的最优路线长度, 必定是多旅行商问题最优路线总长度的下界.

本题的第一问, 要求设计分三组巡视时使总路程最短且各组尽可能均衡的巡视路线. 因此需要将加权图 $G(V,W)$ 中的顶点集 V 划分为 V_1, V_2, V_3, 将 G 分成三个子图 $G(V_1)$, $G(V_2)$ 和 $G(V_3)$ 使得

1. 子顶点集中都包含顶点 O: $O \in V_i$, $i = 1,2,3$;

2. 子顶点集中包含了 V 中所有顶点: $\cup V_i = V$;

3. 子图的最小 Hamilton 回路长度总和最小化: $\min C_\Sigma = \min \sum C_i$, 其中 C_i 为子图 $G(V_i)$ 中最小 Hamilton 回路的总权 (路径长度).

4. 子图的最小 Hamilton 回路长度均衡化: $\min\{C_{\max} - C_{\min}\}$, 其中 $C_{\max}$ 和 $C_{\min}$ 分别为 C_i, $i = 1,2,3$ 中的最大和最小值.

显然目标3和目标4是矛盾的, 也就是不可能同时达到最小. 因此具体求解时, 应两者兼顾, 用多目标的方法处理. 一种简单合理的考虑是转换成一个目标函数 $\min\{C_{\max}\}$.

对于顶点集 V 的划分, 可以利用原图的最小生成树, 从县城出发的最短路生成树, 或者原图的单旅行商路线等为依据来划分. 因为它们都是在某种指示下的最优解, 而这类指标与原题的要求又相当接近. 这里我们选用从县城出发的最短路生成树为依据, 来对顶点集进行划分. 通过图论工具箱中的graphshortestpath函数可以很方便的求出以 O 为单点源到达各点的最短路径和最短路径长度, 从而生成图26所示的从县城出发的最短路生成树, 具体程序见代码14. 图26可以看

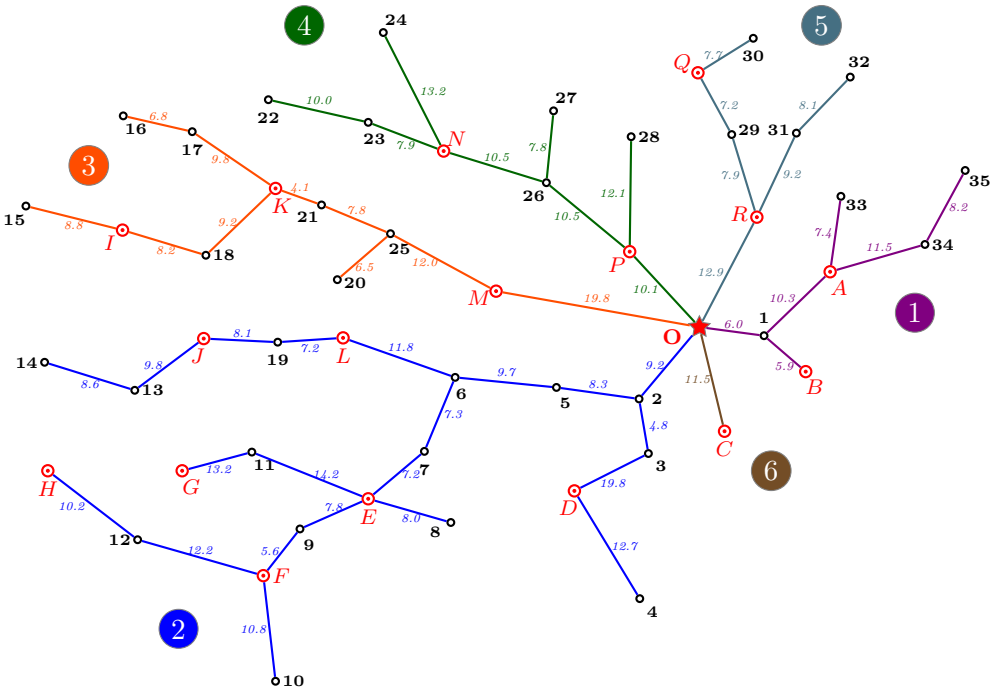
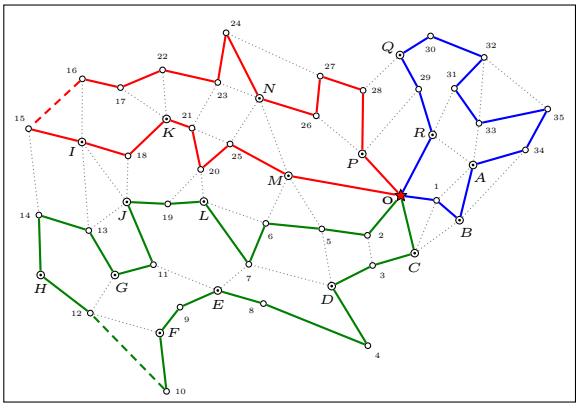


图 26: 从县城出发的最短路生成树

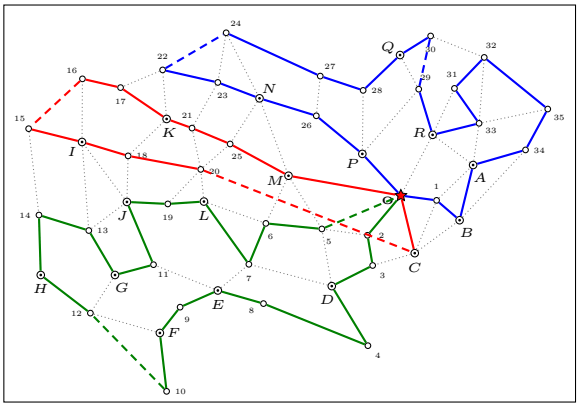
作是从 O 出发的 6 条树干构成的树. 根据上述的四个基本分组原则, 我们给出基于 6 条树干的四种分组方案 (见图27):

- 分组方案 1: (❶ ❺), (❷ ❻), (❸ ❹). 这种分组方案简单, 将两相邻的树干分为一组. 三组的最小 Hamilton 回路长度分别为 125.5, 237.5 和 191.1; 路径总长度为 554.1; 目标函数 $C_{\max} = 237.5$.
- 分组方案 2: (❶ ❹ ❺), (❷), (❸ ❻). 这种分组方案将节点数最多分支❷单独分为一组. 三组的最小 Hamilton 回路长度分别为 217.6, 232.1 和 178.6; 路径总长度为 628.3; 目标函数 $C_{\max} = 232.1$.
- 分组方案 3: (❶ ❺ ❻; 2, 3, 4, C, D), (❷), (❸ ❹). 这种分组方案是在分组方案 1 的基础上的改进方案. 考虑到在分组方案 1 中, 第一组回路长度最短, 第二组回路长度最长, 因此可将第二组中靠近第一组的节点 2, 3, 4, C, D 为划分给第一组. 这种分组方案给出的三组最小 Hamilton 回路长度分别为 188.7, 216.5 和 191.1; 路径总长度为 596.3; 目标函数 $C_{\max} = 216.5$.
- 分组方案 4: (❶ ❹ ❺), (❷; 15, 18), (❸ ❻; 22, 3, 4, 8, 11, 13, D, G). 这种分组方案是在分组方案 2 的基础上的改进方案. 将第三组中靠近第二组的节点 15 和 18 划分给第二组, 并将第一组中的节点 22 和第二组中的节点 3, 4, 8, 11, 13, D, G 划分给第三小组. 这种分组方案给出的三组最小 Hamilton 回路长度分别为 197.6, 203.5 和 200.4; 路径总长度为 607.6; 目标函数 $C_{\max} = 203.5$.

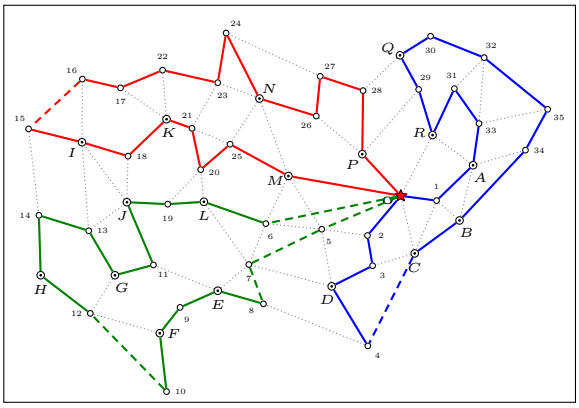
以上四种分组方案的最短巡视路线 (Hamilton 回路) 的求解程序见代码15.



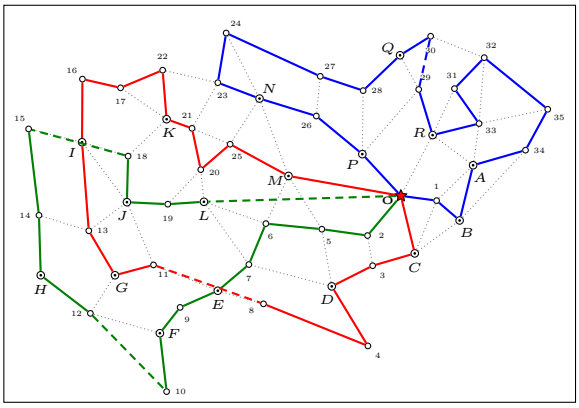
分组方案 1: $C_\Sigma = 554.1$; $C_{\max} = 237.5$



分组方案 2: $C_\Sigma = 628.3$; $C_{\max} = 232.1$



分组 3: $C_\Sigma = 596.3$; $C_{\max} = 216.5$



分组 4: $C_\Sigma = 607.6$; $C_{\max} = 203.5$

图 27: 将顶点集分三组, 四种分组方式的最小 Hamilton 回路

本题的第二问, 添加了巡视组停留时间 $T = 2$ 小时, $t = 1$ 小时及汽车行驶速度为 $v = 35$ km/h 的条件后, 要求在 24 小时内完成巡视的最小分组及相应的最佳巡视路线. 由于原图的单旅行商路线长度是分组巡视总路程的下界, 且我们已经求得单旅商最短路线长度大于 500km. 因此各组花在汽车行驶上的时间之和将超过 14 小时 ($500/35 = 14.3$). 加上在 17 乡 (镇) 和 35 个村的停留时间 $17 \times T + 35 \times t = 17 \times 2 + 35 \times 1 = 69$ 小时. 各组所需时间之各将大于 83 小时. 若分成三组, 就不可能在 24 小时内完成巡视. 因此, 所求的最小分组数为 4.

按照第一问中的分组规则, 设将顶点集 V 划分为四组巡视. C_i 为第 i 组的巡视路线长度; m_i, n_i 分别为该组停留的乡 (镇) 和村数, 则各组所花时间 H_i 为

$$H_i = T \cdot m_i + t \cdot n_i + \frac{C_i}{v} = 2m_i + n_i + \frac{C_i}{35}, \quad i = 1, 2, 3, 4$$

和第一问的情况类似, 所谓最佳的要求仍然是两个目标:

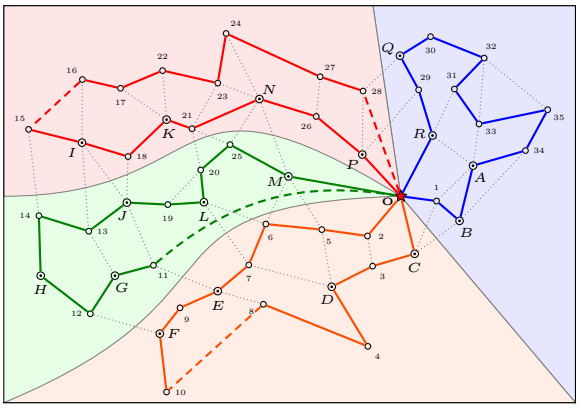
$$\min H_\Sigma = \min \sum H_i \qquad 及 \qquad \min\{H_{\max} - H_{\min}\}$$

我们仍然将以上两个目标转化为一个主要目标 $\min\{H_{\max}\}$ 来考虑. 那么, 乡村数的均衡性和各组路程的均衡性依然是分组的主要依据. 参照第一问的解答和所得结果, 对各组的交界作适当的调整, 可得以下两种分组方案 (见图28):

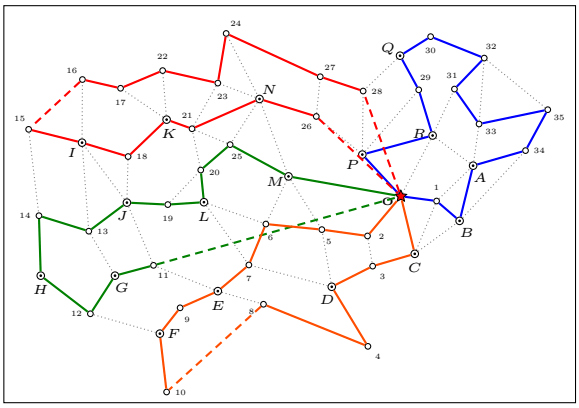
- 分组方案 1: (❶ ❺), (❷ ❻), (❸ ❹), (11, 12, 13, 14, 19, L, H, G, 20, 25, M, J). 这种分组方案主要是将分枝 1, 5 分为一组; 分枝 4 和分枝 3 的部分结点分为一组; 分枝 6 和分枝 2 的部分结点分为一组; 分枝 3 的剩余部分结点和分枝 4 的剩余部分结点分为一组; 这一分组给出的四组巡视时间分别为 19.59, 21.54, 24.14, 21.80 小时; 四组巡视时间总和为 87.07 小时; 目标函数 $H_{\max} = 24.14$.

- 分组方案 2: (❶ ❺; P), (❷ ❻), (❸ ❹), (11, 12, 13, 14, 19, L, H, G, 20, 25, M, J). 分组方案 2 在分组方案 1 基础上进行了微调: 将 P 结点划分给第一组. 这一分组给出的四组巡视时间分别为 22.05, 21.54, 22.14, 21.80 小时; 四组巡视时间总和为 87.53 小时; 目标函数 $H_{\max} = 22.14$.

以上两种分组方案的最短巡视时间的求解程序见代码16.



分组方案 1: $H_\Sigma = 87.07$; $H_{\max} = 24.14$



分组方案 2: $H_\Sigma = 87.53$; $H_{\max} = 22.14$

图 28: 将顶点集分四组, 四种分组方式的最小 Hamilton 回路

本题的第三问, 如果有足够多的巡视人员, 要求出完成巡视的最短时间, 并给出在最短时间限制下的最佳巡视路线. 实际上, 完成巡视的最短时间受到单独巡视离县城最远乡 (镇) 所需时间的制约, 应用graphshortestpath函数可以求得从县城 (节点 O) 到各节点的最短距离. 离县城最远的乡镇为 H 点, 距离为 77.5km, 单独巡视该乡镇所需时间为 $77.5 \times 2/35 + 2 = 6.43$ 小时 (离县城最远的村为 14, 若单独巡视所需时间要小得多). 即使人员足够多, 分组再细, 完成巡视的时间至少需要 6.43 小时. 于是, 问题便转化成在 6.43 小时内完成巡视所需要的最少分组数和巡视方案. 容易证明, 要在 6.43 小时内完成巡视, 有些点 (如 I, J, H) 只能单独作为一组, 时间不允许在别的村停留. 而绝大部分乡村可以和其它乡村分在同一组内, 并在限定时间内完成巡视. 这里我们不再具体讨论如何分组, 有兴趣的读者可以参考 [50], 该文证明了最小分组数为 22.

本题的第四问, 要求在巡视组已定的情况下尽快完成巡视, 讨论参数 T, t 和 v 的改变对最佳巡视路线的影响. 该问题的讨论已经超出了本章的图论范畴, 这里我们不再具体讨论, 有兴趣的读者可以参考出题人评论 [48] 中的相关部分.

代码 12: 自定义函数 road2graph

```
function [w, x,y, n, O] = road2graph(gtype, ifplot)

if nargin<1;                            % 如果没有输入参数
    gtype = 'original'; ifplot = 'plot';
elseif nargin<2;                        % 如果只有一个输入参数
    ifplot = 'plot';
end

nvil  = 35;                             % 村数量
ntown = 18;                             % 镇数量
n = nvil + ntown;                       % 节点数

ID = num2cell(nvil+1:nvil+ntown);       % 镇节点索引
[A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R] = deal(ID{:});% A=36, B=37, ...

%% 村镇节点坐标, 原题并没给出, 这里仅为作图需要
x = [61.7 51.3 52.0 51.3 44.4 35.9 33.3 35.5 22.9 20.9 ... %  1-10
```

```
      18.9  9.4  9.1  1.6  0.0  8.2 13.9 15.1 21.1 26.1 ... % 11-20
      24.7 20.3 28.6 29.9 30.5 43.5 44.1 50.6 59.0 60.8 ... % 21-30
      64.4 68.9 68.1 75.1 78.5 67.2 65.2 58.4 45.9 28.6 ... % 31-35, A-E
      19.9 13.1  1.9  8.1 14.9 20.9 26.5 39.3 34.9 56.3 ... %  F- O
      50.5 56.2 61.1                                     ]'; %  P- R

y = [28.9 23.6 19.0  6.9 24.6 25.4 19.2 13.3 12.7  0.0 ... %  1-10
      19.2 11.8 24.3 26.7 39.7 47.2 45.9 35.6 28.3 33.6 ... % 11-20
      39.8 48.6 46.7 54.2 37.4 41.7 47.7 45.5 45.7 53.7 ... % 21-30
      45.8 50.5 40.5 36.5 42.7 34.2 25.9 20.9 15.9 15.3 ... % 31-35, A-E
       8.8 17.6 17.6 37.7 28.6 41.2 28.7 32.6 44.3 29.6 ... %  F- O
      35.9 50.8 38.8                                     ]'; %  P- R

%% 距离矩阵: w(i,j), i < j 表示 i 和 j 间公路长度
w = zeros(n);

w(1, A) = 10.3; w(1, B) =  5.9; w(1, C) = 11.2; w(1, O) =  6.0;
w(2, 3) =  4.8; w(2, 5) =  8.3; w(2, O) =  9.2;
w(3, C) =  7.9; w(3, D) =  8.2;
w(4, 8) = 20.4; w(4, D) = 12.7;
w(5, 6) =  9.7; w(5, D) = 11.3; w(5, M) = 11.4;

w(6, 7) =  7.3; w(6, L) = 11.8; w(6, M) =  9.5;
w(7, D) = 15.1; w(7, E) =  7.2; w(7, L) = 14.5;
w(8, E) =  8.0;
w(9, E) =  7.8; w(9, F) =  5.6;
w(10,F) = 10.8;

w(11,E) = 14.2; w(11,G) =  6.8; w(11,J) = 13.2;
w(12,F) = 12.2; w(12,G) =  7.8; w(12,H) = 10.2;
w(13,14)=  8.6; w(13,G) =  8.6; w(13,I) = 16.4; w(13,J) = 9.8;
w(14,15)= 15.0; w(14,H) =  9.9;
w(15,I) =  8.8;

w(16,17)=  6.8; w(16,I) = 11.8;
w(17,22)=  6.7; w(17,K) =  9.8;
w(18,I) =  8.2; w(18,J) =  8.2; w(18,K) =  9.2;
w(19,20)=  9.3; w(19,J) =  8.1; w(19,L) =  7.2;
w(20,21)=  7.9; w(20,25)=  6.5; w(20,L) =  5.5;

w(21,23)=  9.1; w(21,25)=  7.8; w(21,K) =  4.1;
w(22,23)= 10.0; w(22,K) = 10.1;
w(23,24)=  8.9; w(23,N) =  7.9;
w(24,27)= 18.8; w(24,N) = 13.2;
w(25,M) = 12.0; w(25,N) =  8.8;

w(26,27)=  7.8; w(26,N) = 10.5; w(26,P) = 10.5;
w(27,28)=  7.9;
w(28,P) = 12.1; w(28,Q) =  8.3;
w(29,P) = 15.2; w(29,Q) =  7.2; w(29,R) =  7.9;
w(30,32)= 10.3; w(30,Q) =  7.7;

w(31,32)=  8.1; w(31,33)=  7.3; w(31,R) =  9.2;
w(32,33)= 19.0; w(32,35)= 14.9;
w(33,35)= 20.3; w(33,A) =  7.4;
w(34,35)=  8.2; w(34,A) = 11.5; w(34,B) = 17.6;

w(A, B) = 12.2; w(A, R) =  8.8;
```

```
w(B, C) = 11.0;
w(I, J) = 15.8;
w(C, O) = 11.5;
w(M, N) = 14.2; w(M, O) = 19.8;

w(O, P) = 10.1; w(O, R) = 12.9;

w = sparse(w');                          % 将 w 转化为下三角稀疏矩阵

%% 如果选项 ifplot 的值为'plot', 则作图
if strcmp(ifplot,'plot')
    [i,j] = find(w>0);
    plot([x(i),x(j)]', [y(i),y(j)]','o:b','linewidth',4);
end

%% 如果选项 gtype 的值为'complete', 则将 w 转化为完全图
if strcmp(gtype,'complete')
   w = graphallshortestpaths(w, 'directed', false);
end
```

代码 13: 图25的单旅行商问题求解脚本 sgltsp

```
%% 获得完全图的距离矩阵 w 和每个节点坐标 (x,y), 并画出原图
[w, x,y] = road2graph('complete', 'plot'); hold on

%% 调用自定义函数 minhamiltonpath 求解最短 Hamilton 路径
[order, totdist] = minhamiltonpath(W);

%% 在当前图形上画出最短 Hamilton 路径
plot(x(order([end 1:end])), y(order([end 1:end])),'r-','linewidth',2)
```

代码 14: 从县城出发的最短路生成树的求解函数 roadtree

```
function [ncomp, icomp] = roadtree(ifplot)

if nargin<1; ifplot = 'plot'; end    % 如果没有指定, 默认画图

%% 获得原图距离矩阵, 每个节点坐标, 节点数及节点 O 的标号
[w, x,y, n, O] = road2graph('original', 'noplot');

%% 求出以 O 为单点源到达各点的最短路径和最短路径长度
[dist, path, pred] = graphshortestpath(w, O, 'directed', false);
I = setdiff(1:n, O);                     % I= 除 O 节点外所有节点
J = pred(I);                             % J=I 中所有节点的前点

%% 构造从节点 O 出发的最短路生成树
tree = zeros(n);
for k = 1:length(I)
    i = I(k); j = J(k);
    tree(i,j) = w(i,j); tree(j,i) =w(j,i);
end

%% 去掉生成树中与 O 节点相关的边, 再用 graphconncomp 求图中的连通分支
tree0 = tree; tree0(:,O) = 0; tree0(O,:) = 0;
[ncomp, icomp] = graphconncomp(sparse(tree0),'directed',false);

%% 画出树, 并对不同连通分支的边着不同颜色
if strcmp(ifplot,'plot')
```

```
    color = {'r','g','b','m','k','c'};
    for k = 1:length(I)
        i = I(k); j = J(k);
        plot([x(i),x(j)], [y(i),y(j)]',['o-', color{icomp(i)}])
        hold on
    end
end
```

代码 15: 顶点集分为三组的最小 Hamilton 回路的求解脚本 group3

```
type = 1;                                   % 分组方式: type = 1, 2, 3, 4

nvil = 35;                                  % 村数量
ntown = 18;                                 % 镇数量
n = nvil + ntown;                           % 节点数
ngroup = 3;                                 % 分组数

ID = num2cell(nvil+1:nvil+ntown);           % 镇节点索引
[A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R] = deal(ID{:});% A=36, B=37, ...

%% 获得完全图的距离矩阵 w 和每个节点坐标 (x,y)
[w, x,y] = road2graph('complete', 'plot');

%% 获得最短路生成树的分枝数 ncomp 和每个节点所属分枝标号 icomp
[ncomp, icomp] = roadtree('noplot');

%% 根据分类方式 type 来选择执行不同的分组程序块
igroup = zeros(n,1);                        % 初始化每个节点的组号
switch type
    case 1 % 分组 1: (❶ ❺), (❷❻), (❸ ❹)
        igroup(icomp==1) = 1; igroup(icomp==5) = 1;
        igroup(icomp==2) = 2; igroup(icomp==6) = 2;
        igroup(icomp==3) = 3; igroup(icomp==4) = 3;
    case 2 % 分组 2: (❶ ❹ ❺), (❷), (❸❻)
        igroup(icomp==1) = 1; igroup(icomp==4) = 1;
        igroup(icomp==5) = 1; igroup(icomp==2) = 2;
        igroup(icomp==3) = 3; igroup(icomp==6) = 3;
    case 3 % 分组 3: (❶ ❺ ❻; 2, 3, 4, D), (❷), (❸ ❹)
        igroup(icomp==1) = 1; igroup(icomp==5) = 1;
        igroup(icomp==6) = 1; igroup(icomp==2) = 2;
        igroup(icomp==3) = 3; igroup(icomp==4) = 3;
        igroup([2 3 D 4])= 1;
    case 4 % 分组 4: (❶ ❹ ❺), (❷; 15, 18), (❸❻; 22, 3, 4, 8, 11, 13, D, G)
        igroup(icomp==1) = 1; igroup(icomp==4) = 1;
        igroup(icomp==5) = 1; igroup(icomp==2) = 2;
        igroup(icomp==3) = 3; igroup(icomp==6) = 3;
        igroup([15 18])  = 2;
        igroup([22 3 4 8 11 13 D G]) = 3;
end

%% 对每一组内的节点, 构造子图 wk, 并求最短 Hamilton 回路
dist = zeros(ngroup,1);                     % 初始化每组回路长度为 0
color = {'r','m','b'};                      % 绘制各组回路的颜色
for k = 1:ngroup
    i = find(igroup==k|igroup==0);          % 找出 k 组中所有节点和 0 点
    wk = w(i,i);                            % 构造子图 wk
    [order, dist(k)] = minhamiltonpath(wk); % 求最短 Hamilton 回路
    order = i(order);                       % 将子图节点序列转化为原图节点序列
```

```
    hold on                                % 在原图上, 画出子图的最短 Hamilton 回路
    plot(x(order([end 1:end])), y(order([end 1:end])), color{k});
end

%% 计算所有组巡视路径总距离 totdist, 和各组中最长巡视路径长度 mindist
totdist = sum(dist)                        % totdist, 即前文所述 C_Σ
maxdist = max(dist)                        % mindist, 即前文所述 C_max
```

代码 16: 顶点集分为四组的最小巡视时间求解脚本 group4

```
nvil = 35;                                 % 村数量
ntown = 18;                                % 镇数量
n = nvil + ntown;                          % 节点数
ngroup = 4;                                % 分组数

ID = num2cell(nvil+1:nvil+ntown);          % 镇节点索引
[A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R] = deal(ID{:});% A=36, B=37, ...

%% 获得完全图的距离矩阵 w 和每个节点坐标 (x,y)
[w, x,y] = road2graph('complete', 'plot');

%% 获得最短路生成树的分枝数 ncomp 和每个节点所属分枝标号 icomp
[ncomp, icomp] = roadtree('noplot');

%% 根据 icomp 来调整分组
igroup = zeros(n,1);
igroup(icomp==1|icomp==5) = 1;
igroup(icomp==4|icomp==3) = 2;
igroup(icomp==2|icomp==6) = 3;
igroup(P) = 1;
igroup([11 12 13 14 19 20 25 M J L H G])=4;

%% 对每一组内的节点, 构造子图 wk, 并求最短 Hamilton 回路和经过的乡 (镇), 村数量
[dist, nv, nt] = deal(zeros(ngroup,1));
color = {'r','m','b','c'};                 % 绘制各组回路的颜色
for k = 1:ngroup
    i = find(igroup==k|igroup==0);         % 找出 k 组中所有节点和 0 点
    nv(k) = sum( (i≤nvil) );               % 计算 k 组中村数量
    nt(k) = sum( (i> nvil) ) - 1;          % 计算 k 组中除 0 节点外的乡 (镇) 数量
    wk = w(i,i);                           % 构造子图 wk
    [order, dist(k)] = minhamiltonpath(wk); % 求最短 Hamilton 回路
    order = i(order);                      % 将子图节点序列转化为原图节点序列
    hold on                                % 在原图上, 画出子图的最短 Hamilton 回路
    plot(x(order([end 1:end])), y(order([end 1:end])), color{k}, ...
        'linewidth',2);
end

%% 计算各组巡视时间 time 和总时间 tottime, 和各组中最长巡视时间 maxdist
time = 2*nt + nv + dist/35                 % 巡视时间 = 2 * 镇数 + 村数 + 车程/速度
tottime = sum(time)                        % totdist, 即前文所述 H_Σ
maxtime = max(time)                        % mindist, 即前文所述 H_max
```

网页排序算法 PageRank

近些年来, 不少数学建模问题可以转化为网络节点的评级和排序问题, 而评级和排序的依据则是依赖于节点自身的各种中心度或其它性质. 有些赛题明确要求参赛者运用网络模型来

对节点评级或排序. 例如 2014 年美国大学生数学建模竞赛 C 题 “Using Networks to Measure Influence and Impact”[51], 该题要求参赛者建立起学者 Paul Erdos 及其超过 500 个共同合作者的合作关系网络, 并据此来找出该网络中最有影响力的人. 另一些赛题看起来并不是一个网络问题, 但仍可以应用网络模型来对个体进行评价或排序. 例如 2014 年美国大学生数学建模竞赛 B 题 “College Coaching Legends”, 该题要求参赛者建立数学模型找出上个世纪三种体育项目中最好的五名大学体育教练. 这是一个典型的评价问题, 常规思路是找出能反映教练水平的因素并量化, 然后确定各自的权重并进行加权, 以此来评价每一个教练, 最后根据评分来对教练排名. 当年几乎所有特等奖论文都是这种思路, 但其中一篇来自 NC School of Science and Mathematics 的特等奖论文 [52] 却另辟蹊径, 作者根据教练们所执教的体育队间的比赛胜负关系, 建立起有向图网络. 每个节点表示一个参赛队, 如果 A 队战胜了 B 队, 则有一条边从 A 指向 B. 然后根据建立的网络求节点的特征向量中心度来确定每个参赛队的水平, 并通过模型把教练水平从参赛队的表现中分离出来, 最终得到教练水平并进行排名. 关于特征向量中心度的定义, 参看第2.5小节 (第8页).

本节中将介绍网络节点评级和排序的一个重要的模型: PageRank 模型 [53]. PageRank 网页排名, 又称网页级别, 是一种由搜索引擎根据网页之间相互的超链计算网页级别的技术. 它是由 Google 的创始人 Larry Page 和 Sergey Mikhaylovich Brin 于 1998 年在斯坦福大学发明的. Google 用 PageRank 来体现网页的相关性和重要性. 在搜索引擎优化操作中, PageRank 是经常被用来评估网页优化的成效因素之一. PageRank 通过网络浩瀚的超链接关系来确定一个页

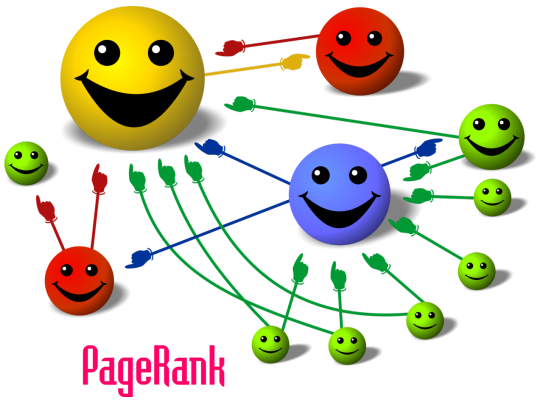


图 29: PageRank 算法的思想 [53]

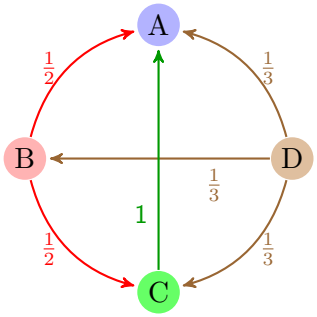


图 30: PageRank 算法的简例

面的等级. Google 把从 A 页面到 B 页面的链接解释为 A 页面给 B 页面投票, Google 根据投票来源 (甚至来源的来源, 即链接到 A 页面的页面) 和投票目标的等级来决定新的等级. 图29反映了 PageRank 算法的思想. 简单的说, 一个高等级的页面可以使其他低等级页面的等级提升.

下面我们以图30所示的简单例子来说明 PageRank 算法, 假设一个由 4 个页面组成的小团体: A, B, C 和 D. 如果所有页面都链向 A，那么 A 的 R(PageRank) 值将是 B, C 及 D 的 PageRank 总和.

$$R(A) = R(B) + R(C) + R(D)$$

继续假设 B 也有链接到 C, 并且 D 也有链接到包括 A 的 3 个页面. 一个页面不能投票 2 次. 所以 B 给每个页面投半票. 以同样的逻辑, D 投出的票只有三分之一算到了 A 的 PageRank 上.

$$R(A) = \frac{R(B)}{2} + \frac{R(c)}{1} + \frac{R(D)}{3} \qquad \text{或} \qquad R(A) = \frac{R(B)}{L_n(B)} + \frac{R(C)}{L_n(C)} + \frac{R(D)}{L_n(D)}$$

其中 $L_n(B)$, $L_n(C)$ 各 $L_n(D)$ 分别表示页面 A, B 和 C 链接的页面数. 最后, $R(A)$ 被换算为一个百分比再乘上一个系数 d. 由于 “没有被任何页面链接的页面” 传递出去的 PageRank 会是 0, 所以, Google 给了每个页面一个最小值 $(1-d)/N$:

$$R(A) = \left(\frac{R(B)}{L_n(B)} + \frac{R(C)}{L_n(C)} + \frac{R(D)}{L_n(D)} + \cdots\right)d + \frac{1-d}{N}$$

其中 N 为页面的总数量; d 被称为阻尼系数 (damping factor), 在 PageRank 算法中 $d = 0.85$, 其意义是，在任意时刻, 用户到达某页面后并继续向后浏览的概率, 该数值是根据上网者使用浏览器书签的平均频率估算而得. $1 - d = 0.15$ 则是用户停止点击, 随机跳到新地址的概率. 我们把上面的例子推广到一般情况, 对于页面 p_i, 其 PageRank 值为

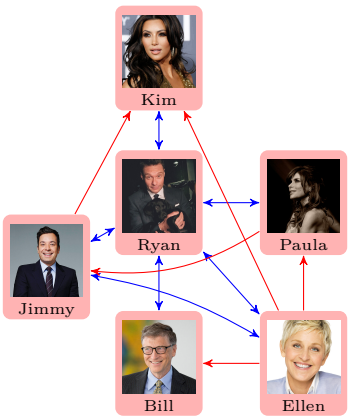
$$R(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{R(p_j)}{L_n(p_j)} \tag{15}$$

其中, $p_1, p_2, \cdots, p_N$ 是被研究的页面, $M(p_i)$ 是链入 p_i 页面的集合, $L_n(p_j)$ 是 p_j 链出页面的数量, 而 N 是所有页面的数量. 式 (15) 的矩阵形式为

$$\mathbf{R} = \begin{bmatrix} \frac{1-d}{N} \\ \vdots \\ \frac{1-d}{N} \end{bmatrix} + d \begin{bmatrix} l_{1,1} & \cdots & l_{1,n} \\ \vdots & \ddots & \vdots \\ l_{n,1} & \cdots & l_{n,n} \end{bmatrix} \mathbf{R}, \qquad \text{其中} \quad \mathbf{R} = \begin{bmatrix} R(p_1) \\ \vdots \\ R(p_N) \end{bmatrix} \tag{16}$$

如果 p_j 不链向 p_i, 则 $l_{i,j}$ 等于 0. 并且对任意页面 j 都标准化有 $\sum_{i=1}^{N} l_{i,j} = 1$. PageRank 技术的主要缺点是旧的页面等级会比新页面高. 因为即使是非常好的新页面也不会有很多外链.

接下来, 我们以如图31所示的名人社交网络 [54] 为例, 计算每个名人主页的 PageRank 值. 名人主页的 PageRank 值可以反映该名人的影响力. 图31中的每条弧表示, 弧尾的名人关注了弧



PageRank
0.3544
0.1526
0.1503
0.1285
0.1071
0.1071

图 31: 名人社交网络及各节点的 PageRank 排序, 素材来源 [54]

头的名人. 比如 Ellen 有一条弧指向 Bill, 表示 Ellen 关注了 Bill. 代码17给出了求解名人社交网络中各节点的 PageRank 值的 Matlab 程序. 求得的结果已经列在了图31中, 从结果中可以看出, Ryan 在名人社交网中的影响力是最大的, Jimmy 其次. Bill 和 Paula 则排在最后, 且他们俩的 PageRank 值相同, 这是由于关注他俩都只有 Ellen 和 Ryan.

代码 17: 名人社交网络中个体的 PageRank 值

```
d = 0.85;                % 阻尼系数
name = {'bill', 'ellen', 'jimmy', 'kim', 'paula', 'ryan'}; % 名人姓名
[bil, ell, jim, kim, pau, rya] = deal(1,2,3,4,5,6);        % 给名人编号
n = length(name);    % 名人数量

link = zeros(n);     % 关系矩阵, 如果j关注了i,  则link(i,j) = 1
link(bil, [rya, ell]) = 1;
link(jim, [rya, pau, ell]) = 1;
link(kim, [jim, rya, ell]) = 1;
link(pau, [rya, ell])=1;
link(rya, [bil, jim, kim, pau, ell]) = 1;
link(ell, [jim, rya]) = 1;
```

```
13
14 Ln = sum(link,1);                          % 关注的人数(出度)
15
16 L = bsxfun(@times, link, 1./Ln);           % L(i,:) = link(i,:)./Ln
17
18 C = (1-d)/n * ones(n,1);                   % C = [(1-d)/n, ..., (1-d)/n]^T
19 I = eye(n);                                % 单位阵
20
21 R = (I - d*L)\C        % R = C + dLR  ⟹  R = (I - dL)^{-1}C
```

代码17中有两个函数需要额外说明: 第3行中的函数deal是用来批量赋值的, 第3行程序的作用是把 bil, ell, jim, kim, pau, rya 分别赋值为 1 到 6; 第16行中的函数bsxfun是用来将link的每一行都点乘1./Ln向量.

参考文献

[1] James Joseph Sylvester. Chemistry and algebra. *Nature*, 17:284, 1878.

[2] Graph theory. *Wikipedia, the free encyclopedia*, February 2015. `http://en.wikipedia.org/wiki/Graph_theory`.

[3] N. Biggs, E. Lloyd, and Wilson. *Graph Theory*. Oxford University Press, 1986.

[4] Mark R. Keen. The knight's tour. `http://www.markkeen.com/knight`.

[5] Daniel Garber. Geometry and Monadology: Leibniz' s Analysis Situs and Philosophy of Space, by Vincenzo De Risi. *Mind*, 119(474):472–478, 2010.

[6] A. L. Cauchy. Recherche sur les polyèdres-premier mémoire. *Journal de l'Ecole Polytechnique. v9*, pages 66–86.

[7] Icosian game. *Wikipedia, the free encyclopedia*, June 2014. `http://en.wikipedia.org/wiki/Icosian_game`.

[8] Hamiltonian path problem. *Wikipedia, the free encyclopedia*, March 2015. `http://en.wikipedia.org/wiki/Hamiltonian_path_problem`.

[9] Arthur Cayley. On the theory of the analytical forms called trees. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 13(85):172–176, 1857.

[10] George Pólya. *Wikipedia, the free encyclopedia*, May 2015. `http://en.wikipedia.org/wiki/George_Polya`.

[11] Nicolaas Govert de Bruijn. *Wikipedia, the free encyclopedia*, May 2015. `http://en.wikipedia.org/wiki/Nicolaas_Govert_de_Bruijn`.

[12] A Cayley. Ueber die analytischen figuren, welche in der mathematik bäume genannt werden und ihre anwendung auf die theorie chemischer verbindungen. *Berichte der deutschen Chemischen Gesellschaft*, 8(2):1056-1059, 1875.

[13] Seven Bridges of Königsberg. *Wikipedia, the free encyclopedia*, March 2015. `http://en.wikipedia.org/wiki/Seven_Bridges_of_Konigsberg`.

[14] Four color theorem. *Wikipedia, the free encyclopedia*, April 2015. `http://en.wikipedia.org/wiki/Four_color_theorem`.

[15] MCM: The Mathematical Contest in Modeling. `http://www.comap.com/undergraduate/contests/mcm/previous-contests.php`.

[16] Adrian Bondy and U. S. R. Murty. *Graph Theory*. Springer, New York, 2008 edition edition, November 2010.

[17] Network theory. *Wikipedia, the free encyclopedia*, March 2015. `http://en.wikipedia.org/wiki/Network_theory`.

[18] Centrality. *Wikipedia, the free encyclopedia*, April 2015. `http://en.wikipedia.org/wiki/Centrality`.

[19] PageRank. *Wikipedia, the free encyclopedia*, May 2015. `http://en.wikipedia.org/wiki/PageRank`.

[20] Social Network Analysis, A Brief Introduction. `http://orgnet.com/sna.html`.

[21] MATLAB & Simulink - MathWorks. Network Analysis and Visualization. http://www.mathworks.com/help/bioinfo/network-analysis-and-visualization.html.

[22] 司守奎, 孙玺菁. 数学建模算法与应用. 国防工业出版社, 第 1 版 edition, August 2011.

[23] MIT Strategic Engineering Research Group. Matlab Tools for Network Analysis. http://strategic.mit.edu/downloads.php?page=matlab_networks.

[24] Gergana. Octave networks toolbox. https://github.com/aeolianine/octave-networks-toolbox.

[25] NetworkX, High-productivity software for complex networks. https://networkx.github.io.

[26] Shortest path problem. *Wikipedia, the free encyclopedia*, April 2015. http://en.wikipedia.org/wiki/Shortest_path_problem.

[27] Dijkstra's algorithm. *Wikipedia, the free encyclopedia*, May 2015. http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.

[28] Floyd – Warshall algorithm. *Wikipedia, the free encyclopedia*, May 2015. http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm.

[29] Minimum spanning tree. *Wikipedia, the free encyclopedia*, May 2015. http://en.wikipedia.org/wiki/Minimum_spanning_tree.

[30] Prim's algorithm. *Wikipedia, the free encyclopedia*, May 2015. http://en.wikipedia.org/wiki/Prim%27s_algorithm.

[31] Kruskal's algorithm. *Wikipedia, the free encyclopedia*, April 2015. http://en.wikipedia.org/wiki/Kruskal%27s_algorithm.

[32] Maximum flow problem. *Wikipedia, the free encyclopedia*, April 2015. http://en.wikipedia.org/wiki/Maximum_flow_problem.

[33] Ford – Fulkerson algorithm. *Wikipedia, the free encyclopedia*, April 2015. http://en.wikipedia.org/wiki/Ford-Fulkerson_algorithm.

[34] Edmonds – Karp algorithm. *Wikipedia, the free encyclopedia*, May 2015. http://en.wikipedia.org/wiki/Edmonds-Karp_algorithm.

[35] Breadth-first search. *Wikipedia, the free encyclopedia*, May 2015. http://en.wikipedia.org/wiki/Breadth-first_search.

[36] Algorithm Implementation/Graphs/Maximum flow/Edmonds-Karp. *Wikipedia, the free encyclopedia*, May 2015. http://en.wikibooks.org/wiki/Algorithm_Implementation/Graphs/Maximum_flow/Edmonds-Karp.

[37] Minimum-cost flow problem. *Wikipedia, the free encyclopedia*, June 2015. http://en.wikipedia.org/wiki/Minimum-cost_flow_problem.

[38] Algorithme de Busacker et Gowen. *Wikipedia, the free encyclopedia*, February 2015. http://fr.wikipedia.org/wiki/Algorithme_de_Busacker_et_Gowen.

[39] Hamiltonian path. *Wikipedia, the free encyclopedia*, May 2015. https://en.wikipedia.org/wiki/Hamiltonian_path.

[40] Travelling salesman problem. *Wikipedia, the free encyclopedia*, July 2015. https://en.wikipedia.org/wiki/Travelling_salesman_problem.

[41] Gábor Pataki. Teaching integer programming formulations using the traveling salesman problem. *SIAM review*, 45(1):116–123, 2003.

[42] 勇肖华. 大学生数学建模竞赛指南. 电子工业出版社, 第 1 版 edition, April 2015.

[43] MathWorks. Travelling salesman problem. http://www.mathworks.com/help/optim/ug/travelling-salesman-problem.html.

[44] David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems. *Mathematical programming*, 97(1-2):91–153, 2003.

[45] Simulated annealing, December 2014. Available at http://en.wikipedia.org/wiki/Simulated_annealing[Accessed December 20, 2014].

[46] 丁颂康. 灾情巡视路线. 全国大学生数模竞赛题目汇编 *(1992-2000)*, page 11. http://www.mcm.edu.cn/upload_cn/node/1/SkAh1A7Q6f2dd01e58aa621f920e79f45cd5d255.pdf.

[47] 杨庭栋, 李晓涛, 郑长江, and 赵静. 最佳灾情巡视路线的数学模型. 数学的实践与认识, 20(1):50–59, 1999.

[48] 丁颂康. 灾情巡视的最佳路线. 数学的实践与认识, 29(1):74–78, 1999.

[49] Vehicle routing problem. *Wikipedia, the free encyclopedia*, June 2015. https://en.wikipedia.org/wiki/Vehicle_routing_problem.

[50] 俞文 (鱼此). 多旅行商路线的几个问题. 数学的实践与认识, 29(1):79–86, 1999.

[51] Comap. 2014 MCM/ICM Problems. http://www.comap.com/undergraduate/contests/mcm/contests/2014/problems/.

[52] Tian-Shun Jiang, Zachary Polizzi, and Christopher Yuan. A Networks and Machine Learning Approach to Determine the Best College Coaches of the 20th-21st Centuries. *arXiv:1404.2885 [cs, stat]*, April 2014. http://arxiv.org/abs/1404.2885.

[53] PageRank. *Wikipedia, the free encyclopedia*, May 2015. http://en.wikipedia.org/wiki/PageRank.

[54] Math Movement. Celebrity networking. http://sites.davidson.edu/mathmovement/celebrity-networking/.