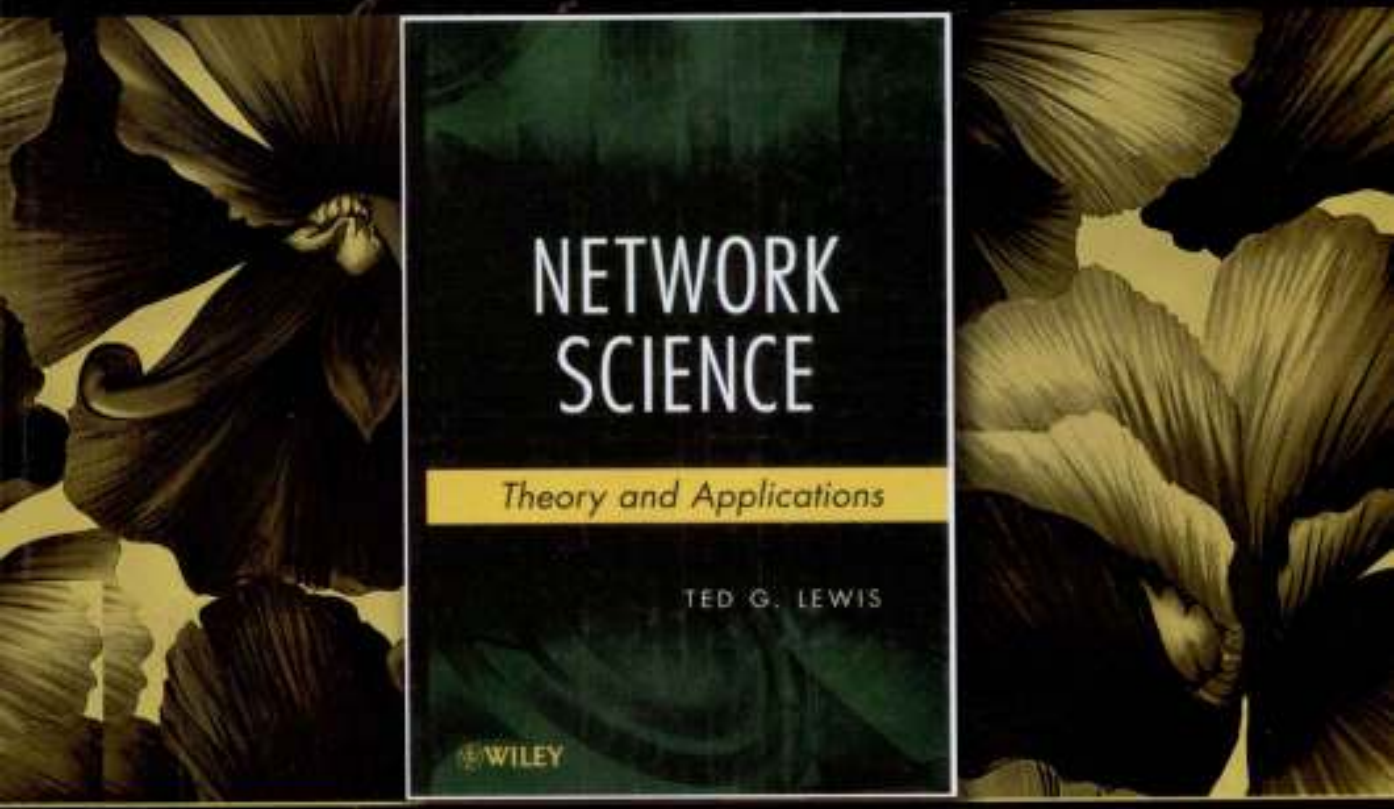
计 算 机 科 学 丛 书

网络科学
原理与应用

（美）Ted G. Lewis 著　陈向阳 巨修练 等译

Network Science
Theory and Applications



机械工业出版社
China Machine Press



网络科学 原理与应用

Network Science Theory and Applications

网络科学作为一门新兴的学科越来越引入瞩目。网络科学能帮助读者设计更快、更有弹性的通信网络；能用于调整电力网络、电信网络和飞行航线等基础设施系统；可以为市场动态建模；能帮助理解生物系统中的同步；能用于分析人与人之间的社会互动……

这是第一本全面审视新出现的网络科学的论著。书中研究了各种网络——规则网络、随机网络、小世界网络、影响网络、无标度网络和社会网络等，并将网络过程和行为应用于涌现、传染病、同步和风险方面。本书的独特之处在于将跨计算机科学、生物学、物理学、社会网络分析、经济学和市场营销等多学科的专业概念整合了起来。

本书为网络科学领域提供了全新的理解和阐释，是研究人员、专业人员以及工程、计算、生物领域的技术人员不可缺少的参考资料，也可以作为相关专业高年级本科生和研究生教材。

作者简介

Ted G. Lewis 博士是美国加州蒙特雷海军研究生院的计算机科学教授。他有丰富的工作阅历和广泛的学术研究，作为美国电气和电子工程师协会(IEEE)计算机协会的会员，他还担任《IEEE软件》和《计算机》杂志的总编，并已经编著出版了30余本著作。他曾任伊士曼柯达公司(Eastman Kodak Company)数字策略的副总裁。

客服热线：(010) 68378991, 88361066
购书热线：(010) 68326294, 68379649, 68995259
投稿热线：(010) 88379604
读者信箱：hzjsj@hzbook.com

华章网站 http://www.hzbook.com

网上购书：www.china-pub.com

WILEY
www.wiley.com

上架指导：计算机 网络科学

ISBN 978-7-111-35966-1

9 787111 359661

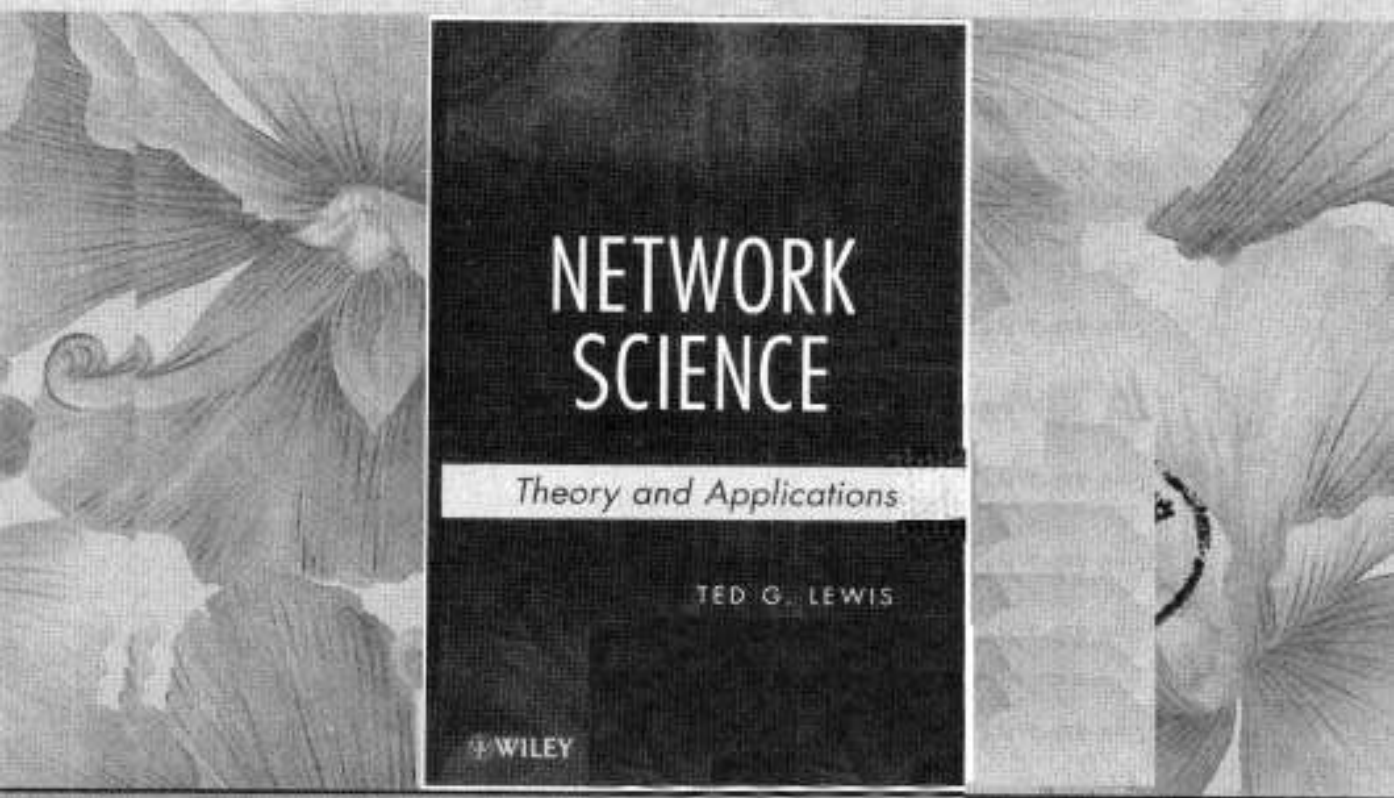
定价：55.00元

计 算 机 科 学 丛 书

网络科学
原理与应用

（美）Ted G. Lewis 著　陈向阳 巨修练 等译

Network Science
Theory and Applications



机械工业出版社
China Machine Press

网络科学作为一门不断发展的新兴交叉学科，以图论为基础，在不断吸收了其他专业的最新成果后终于以一门独立的学科出现。它不仅可以作为新兴的网络工程、网络安全等理工科专业的理论基础，还可以作为网络经济、网络营销、社会科学研究的基本理论，而且是公共关系定量分析的重要工具之一。全书很好地将理论和应用相结合，首先系统地介绍了网络科学的发展历程，然后进一步阐述了大量网络分析与规划的实例，内容覆盖了社会关系网、生物网络、电力网络、病毒传播网、互联网等内容，试图为读者描述一种广义的网络模型。

本书适用于网络工程、网络通信、网络安全、网络营销、网络信息传媒、公共关系、应用数学等相关课程的高年级本科生和研究生教材或参考书，也是研究人员、专业人员以及工程、计算、生物、化学领域的技术人员非常好的参考资料。

Ted G. Lewis: *Network Science: Theory and Applications* (ISBN 978-0-470-33188-0).
Authorized translation from the English language edition published by John Wiley & Sons, Inc.
Copyright © 2009 by John Wiley & Sons, Inc.
All rights reserved.

本书中文简体字版由约翰－威利父子公司授权机械工业出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

封底无防伪标均为盗版
版权所有，侵权必究
本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2011-1457

图书在版编目（CIP）数据

网络科学：原理与应用/（美）路易斯（Lewis，T. G.）著；陈向阳等译．—北京：机械工业出版社，2011.9
（计算机科学丛书）
书名原文：Network Science: Theory and Applications

ISBN 978-7-111-35966-1

Ⅰ. 网…　Ⅱ. ①路…　②陈…　Ⅲ. 计算机网络－研究　Ⅳ. TP393

中国版本图书馆 CIP 数据核字（2011）第 194814 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码　100037）
责任编辑：刘立卿
北京市荣盛彩色印刷有限公司印刷
2011 年 10 月第 1 版第 1 次印刷
185mm×260mm · 22 印张
标准书号：ISBN 978-7-111-35966-1
定价：55.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991；88361066
购书热线：（010）68326294；88379649；68995259
投稿热线：（010）88379604
读者信箱：hzjsj@hzbook.com

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自 1998 年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage 等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出 Andrew S. Tanenbaum，Bjarne Stroustrup，Brain W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www. hzbook. com
电子邮件：hzjsj@hzbook. com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街 1 号
邮政编码：100037



华章教育
华章科技图书出版中心

译者序

Network Science: Theory and Applications

“我们被困在无法逃避的相互关系网络中，任何事情，如果直接地影响了一个人，就会间接地影响到所有人。”

——马丁·路德·金

目前，网络科学在不断汲取各学科最新成果的基础上继续深入发展、完善，其理论研究方法成为广泛的交叉科学的一种强有力的思想方法。2009 年 7 月 24 日出版的《科学》杂志刊登专题——“复杂系统与网络”（Complex Systems and Networks），充分地表明网络科学进一步向众多学科渗透并向应用发展。从网络科学的角度来看，无论是自然界还是人类社会，网络都无处不在，从而深刻而广泛地影响着人们的日常生活和科学技术等各种活动。因此，利用网络科学，可以探讨自然界和人类社会的各种各样的复杂系统。网络科学正在与众多新兴科学相互交融和推动，它提供了新的科学发展观和方法论，使决定性与随机性、有序性与无序性、复杂性与简单性，达到了和谐统一，人类的认识产生了新的飞跃，成为人们认识客观世界的工具。在网络科学的思想、理论与方法的大框架下，无论从微观层次，还是宏观和宇宙观层次，人们都可以从全新的网络的角度、观点和方法来探讨世界万物的复杂性问题。

为了弥补和丰富国内有关网络科学方面的教材，同时也为了科研需要，我们组织翻译了美国蒙特雷海军研究生院的计算机科学教授特德·刘易斯（Ted Lewis）博士的《网络科学：原理与应用》这本书。本书由陈向阳、巨修练负责翻译，参加翻译的人员还包括徐清、陈晓明、徐茜、蹇贝、孙金余、费滕、李亚玲、孙克华、於照等。机械工业出版社刘立卿编辑、王春华编辑等在审稿过程中做了大量辛苦的工作，在此特别致以衷心的感谢！本书的翻译得到了绿色化工过程教育部重点实验室及武汉工程大学优秀学术著作出版资助项目的资助。

在翻译时，由于本书涉及的领域广泛，具体包括应用数学、计算机科学、生物学、物理、化学、医学、社会学、军事等专业，所以挑战在所难免。虽然我们参阅了大量复杂网络的相关译文资料——特别是中科院方锦清对本书的评价、武汉大学陆俊安等发布的网络科学博文，而且花了大量的时间尽力使这本译著完美，但由于自身水平和专业局限，译文中仍会有错误和不到之处。读者在使用本书时若遇到问题或者有好的建议，敬请联系我们（xychensun@ yahoo. com. cn），以期共同做好网络科学的学习和研究。

译者

2011 年 3 月

在我写本书前言时，“网络科学”可能还不够成熟，目前就宣布将图论、控制论和交叉学科应用结合成一门“科学”还为时过早。的确，我的许多同事也表达了强烈的不同意见——他们认为网络科学是一种时尚，甚至是编造出来的东西。带着这种担心，从2006年起我开始写一系列有关无标度网络、小世界和自同步网络的论文。这一系列论文所表达出的粗糙理念逐渐演变成了目前捧在你手中的这本书。像大多数首次尝试一样，本书不可能没有缺陷。我写作本书是出自对科学的热衷——希望它能成为无偏见的、探索思想的有用资料。或许将来会建立一门几乎涉及每个学科、工程、医学和社会科学领域研究的新的网络科学。

只有时间才能证明这种将我们所知的新的网络科学编译成卷的初步尝试是否成功——而本书正是付诸实践之作！但是写作略微超前的主题并不是我的长处。风险在于选择了我称之为**网络科学**的主题。很显然，我们必须概括如 Adamic、Albert、Barabasi、Barrat、Bollobas、Erdos、Granovetter、Kephart、Lin、Liu、Mihail、Milgram、Molloy、Moore、Newman、Pastor-Satorras、Renyi、Strogatz、Tadic、Wang、Watts、Weigt、White、Zhang 和 Zhu 等先驱们的工作。本书共13章，我在前6章对上述这些作者的工作进行了概括。这些章节从图论基础领域，再到现代网络定义展开。最著名的话题——规则、随机、无标度和小世界网络都有整章的叙述。因此，本书前半部分是沿着发明者们绚丽的足迹追踪网络科学的发展。

我的第二个目的是在先驱们已知和已发表的内容基础之上添加新的内容。这里的风险在于我的自命不凡：大胆地假定已知这种新的领域会朝哪个方向发展。第7章介绍网络的新的自组织原则，并演示如何定制设计任意度序列分布的网络。第8章是将 Z. Wang、Chakrabarti、C. Wang 和 Faloutsos 的卓越工作尝试扩展到令人激动的为互联网设计的抗原对策中。这种工作可以用来解释人类传染病以及席卷整个互联网的传染病毒。第9章将 Watts 的早期工作提高到一个新的水平——强调网络同步仅仅是线性系统稳定性的特殊例子。简单的特征值工具可以用于决定几乎任意线性网络的稳定性和同步性。第10章几乎是全新材料，说明为了使社会网络达成共识需要满足什么条件。事实证明，群组达成共识并不容易！第11章是以 Waleed Al-Mannai 的博士论文为基础构建的，他形式化并扩展了我在网络风险方面的研究工作。Al Mannai 的理论用来衡量关键基础系统并保护它们免受自然和人为（人造）的攻击，这对美国本土安全的实践已经有了深刻的影响。第12章是对商业模型的探索——将著名的 Schumpeter **创造性破坏**过程与涌现过程关联起来，并将 Bass-Fisher-Pry 方程映射到网络上。验证 Bass-Fisher-Pry 方程可用于网络是值得欣慰的，但是更进一步，我还将演示如何将这些经典模型扩展应用到多产品市场和寡头垄断上。最后，第13章完全体现了网络科学中的最前沿内容。这一章向读者介绍激动人心的新领域——蛋白质表达网络，并就读者需要考虑的一些新的方向提供一些建议。

一般读者可以很容易地跳过有关数学内容，并仍能从将网络应用到各种学科（包括计算机科学与工程、商务、公共卫生（传染病）、互联网病毒防范、社会网络行为、生物学和物理学等）中搜集到很多信息。更加专业的读者和教师可能想要使用由出版商和作者提供的软件工具（tedglewis@ friction-

free-economy. com)。其中包括5个Java应用程序：Network. jar用来探索各类网络并用各种涌现过程实验；Influence. jar用于研究影响网络和社会网络分析；NetworkAnalysis. jar用来研究网络脆弱性和攻防双方的网络风险问题；NetGain. jar用于商业建模；BioNet. jar用于生物网络，特别是蛋白质表达网络。可执行代码和源代码都是以开源软件的形式提供的。如果你在大学课程中想要以指导教师的身份分发这些材料，可从出版商的Web站点下载教师手册。

本书只起到抛砖引玉的作用，还留下了许多未回答的问题。我希望本书能给新一代的研究人员及其研究工作带来灵感。如果我是正确的，那么术语“网络科学”在将来的十年就不会再存在争议。当然这留给读者作为练习！

我要感谢Steve Lieberman，他认真地对公式进行了排版并详细地审阅数遍草稿。还要感谢Rudy Darken和Tom Mackin，他们对我的思考过程带来影响，并帮助我纠正了几个错误构思。Waleed Al Mannai对第11章有很大的影响，并间接地对整本书产生了影响。非常高兴能与这些同事一起合作，历时3年完成了这本书的写作。

Ted G. Lewis

2008年3月22日

出版者的话
译者序
前 言

第1章 网络科学的起源 …… 1
1.1 什么是网络科学 …… 4
1.2 网络科学简史 …… 5
1.2.1 网前阶段（1736—1966） …… 5
1.2.2 中期网络阶段（1967—1998） …… 7
1.2.3 现代阶段（1998—） …… 9
1.3 总则 …… 12

第2章 图 …… 15
2.1 图的集合论定义 …… 16
2.1.1 节点、链路和映射函数 …… 16
2.1.2 节点度和 hub …… 17
2.1.3 路径和回路 …… 18
2.1.4 连通性和组件 …… 18
2.1.5 直径、半径和中心性 …… 19
2.1.6 介数和紧度 …… 20
2.2 图的矩阵代数定义 …… 21
2.2.1 连接矩阵 …… 21
2.2.2 邻接矩阵 …… 22
2.2.3 拉普拉斯矩阵 …… 23
2.2.4 路径矩阵 …… 23
2.3 哥尼斯堡七桥图 …… 25
2.3.1 欧拉路径和欧拉回路 …… 25
2.3.2 哥尼斯堡七桥问题的正式定义 …… 25
2.3.3 欧拉解 …… 26
2.4 图的谱属性 …… 27
2.4.1 谱半径 …… 28
2.4.2 谱隙 …… 29
2.5 图的类型 …… 30
2.5.1 杠铃形、线形和环形图 …… 30
2.5.2 结构化图与随机图 …… 33
2.5.3 k－规则图 …… 34
2.5.4 图密度 …… 35
2.6 拓扑结构 …… 35
2.6.1 度序列 …… 36
2.6.2 图的熵 …… 36
2.6.3 无标度拓扑 …… 37
2.6.4 小世界拓扑 …… 38
2.7 软件中的图实现 …… 41
2.7.1 Java 节点和链路 …… 41
2.7.2 Java 网络 …… 42
练习 …… 45

第3章 规则网络 …… 47
3.1 直径、中心性和平均路径长度 …… 48
3.2 二叉树网络 …… 53
3.2.1 二叉树网络的熵 …… 54
3.2.2 二叉树网络的路径长度 …… 55
3.2.3 二叉树网络的链路效率 …… 56
3.3 超环形网络 …… 56
3.3.1 超环形网络的平均路径长度 …… 58
3.3.2 超环形网络的链路效率 …… 59
3.4 超立方网络 …… 59
3.4.1 超立方网络的平均路径长度 …… 62

3.4.2 超立方网络的链路效率 …… 63
练习 …… 64

第4章 随机网络 …… 65

4.1 随机网络的生成 …… 66
4.1.1 Gilbert 随机网络 …… 66
4.1.2 Erdos-Renyi 随机网络 …… 68
4.1.3 锚定随机网络 …… 70
4.2 随机网络的度分布 …… 71
4.3 随机网络的熵 …… 74
4.3.1 随机网络熵的建模 …… 74
4.3.2 随机网络的平均路径长度 …… 75
4.3.3 随机网络的聚类系数 …… 77
4.3.4 随机网络的链路效率 …… 78
4.4 随机网络的属性 …… 78
4.4.1 随机网络的直径 …… 79
4.4.2 随机网络的半径 …… 79
4.4.3 利用 Java 计算紧度 …… 80
4.4.4 随机网络中的紧度 …… 81
4.5 随机网络中的弱联系 …… 83
4.6 规则网络的随机性 …… 84
4.7 分析 …… 85
练习 …… 86

第5章 小世界网络 …… 87

5.1 生成一个小世界网络 …… 89
5.1.1 Watts-Strogatz (WS) 过程 …… 89
5.1.2 一般的 WS 过程 …… 91
5.1.3 小世界网络的度序列 …… 92
5.2 小世界网络属性 …… 94
5.2.1 熵与重联概率 …… 95
5.2.2 熵与密度 …… 97
5.2.3 小世界网络的路径长度 …… 98
5.2.4 小世界网络的聚类系数 …… 100
5.2.5 小世界中的紧度 …… 102
5.3 相变 …… 103
5.3.1 路径长度和相变 …… 103
5.3.2 材料中的相变 …… 104
5.4 小世界网络中的导航 …… 105
5.5 小世界网络中的弱联系 …… 112
5.6 分析 …… 113
练习 …… 115

第6章 无标度网络 …… 116

6.1 生成一个无标度网络 …… 118
6.1.1 Barabasi-Albert (BA) 网络 … 119
6.1.2 生成 BA 网络 …… 121
6.1.3 无标度网络幂律分布 …… 122
6.2 无标度网络的属性 …… 124
6.2.1 BA 网络熵 …… 124
6.2.2 hub 度与密度对应关系 …… 126
6.2.3 BA 网络平均路径长度 …… 127
6.2.4 BA 网络紧度 …… 130
6.2.5 无标度网络聚类系数 …… 131
6.3 无标度网络中的导航 …… 132
6.3.1 最大度导航与密度对应关系 … 133
6.3.2 最大度导航与 hub 度的对应关系 …… 134
6.3.3 在无标度 Pointville 网络中的弱联系 …… 135
6.4 分析 …… 136
6.4.1 熵 …… 136
6.4.2 路径长度和通信 …… 137
6.4.3 聚类系数 …… 138
6.4.4 hub 度 …… 140
练习 …… 140

第7章 涌现 …… 142

7.1 什么是网络涌现 …… 143
7.1.1 开环涌现 …… 144
7.1.2 反馈循环涌现 …… 145
7.2 科学中的涌现 …… 145
7.2.1 社会科学中的涌现 …… 146
7.2.2 物理科学中的涌现 …… 146
7.2.3 生物中的涌现 …… 146
7.3 遗传进化 …… 147

7.3.1 hub 涌现 …… 147
7.3.2 聚类涌现 …… 151
7.4 设计者网络 …… 153
7.4.1 度序列涌现 …… 155
7.4.2 生成给定的度序列的网络 …… 157
7.5 排列网络涌现 …… 160
7.5.1 排列微规则 …… 161
7.5.2 排列和聚类系数 …… 162
7.6 涌现的一个应用 …… 166
7.6.1 随机排列的链路优化 …… 167
7.6.2 确定性排列的优化 …… 169
7.6.3 最小长度涌现模型 …… 169
7.6.4 二维布局 …… 170
练习 …… 171

第8章 传染病 …… 173

8.1 传染病模型 …… 174
8.1.1 Kermack-McKendrick 模型 …… 175
8.1.2 传染病阈值 …… 176
8.1.3 易感－感染－消亡（SIR）模型 …… 177
8.1.4 结构化网络峰值感染密度 … 179
8.1.5 易感－感染－易感（SIS）传染病 …… 181
8.2 网络中持续稳定的传染病 …… 182
8.2.1 随机网络传染病阈值 …… 183
8.2.2 一般网络中的传染病阈值 … 185
8.2.3 一般网络中的固定点感染密度 …… 190
8.3 网络传染病仿真软件 …… 191
8.4 对策 …… 192
8.4.1 对策的算法 …… 193
8.4.2 接种策略对策 …… 193
8.4.3 Java 抗原仿真 …… 195
练习 …… 197

第9章 同步 …… 199

9.1 同步或不同步 …… 200
9.1.1 混沌映射 …… 200
9.1.2 网络稳定性 …… 201
9.2 蟋蟀社会网络 …… 204
9.2.1 蟋蟀社会网络的同步性质 …… 205
9.2.2 更加通用的模型：Atay 网络 …… 210
9.2.3 Atay 网络的稳定性 …… 212
9.3 基尔霍夫网络 …… 216
9.3.1 基尔霍夫网络模型 …… 217
9.3.2 基尔霍夫网络的稳定性 …… 219
9.4 Pointville 电网 …… 222
练习 …… 224

第10章 影响网络 …… 226

10.1 对 buzz 的剖析 …… 227
10.1.1 buzz 网络 …… 228
10.1.2 buzz 网络仿真器 …… 229
10.1.3 buzz 网络的稳定性 …… 230
10.2 社会网络的有用性 …… 232
10.2.1 两方谈判 …… 233
10.2.2 I-nets 状态方程 …… 234
10.2.3 I-nets 的稳定性 …… 236
10.2.4 I-nets 的共识 …… 236
10.2.5 计算影响的 Java 方法 …… 237
10.3 I-nets 中的冲突 …… 239
10.3.1 冲突度 …… 239
10.3.2 计算冲突度的 Java 方法 … 241
10.4 命令层次结构 …… 241
10.5 I-nets 中的有用性涌现 …… 243
10.5.1 加权涌现 …… 244
10.5.2 加权涌现的 Java 方法 …… 245
10.5.3 加权涌现的稳定性 …… 247
10.5.4 链路涌现 …… 249
练习 …… 249

第11章 脆弱性 …… 251

11.1 网络风险 …… 253
11.1.1 将节点作为目标 …… 254

11.1.2 将链路作为目标 …… 255
11.2 关键节点分析 …… 255
11.2.1 杠铃模型 …… 256
11.2.2 网络风险最小化 …… 258
11.2.3 指数成本模型 …… 262
11.2.4 攻击者-防御者模型 …… 262
11.2.5 Java 军备竞赛方法 …… 268
11.3 博弈论的考虑 …… 272
11.4 一般的攻击者-防御者网络风险问题 …… 274
11.5 关键链路分析 …… 275
11.5.1 链路弹性 …… 276
11.5.2 链路弹性模型 …… 278
11.5.3 流弹性 …… 280
11.5.4 流启发式的 Java 方法 …… 282
11.5.5 网络流资源分配 …… 283
11.5.6 结构化网络中的最大流量 … 285
11.6 基尔霍夫网络的稳定性弹性 … 287
练习 …… 288

第 12 章 NetGain 网络 …… 290

12.1 经典扩散方程 …… 292
12.1.1 市场扩散方程 …… 292
12.1.2 简单 NetGain 网络 …… 294
12.2 多产品网络 …… 296
12.3 NetGain 网络涌现的 Java 方法 …… 299
12.4 新兴市场网络 …… 300
12.4.1 新生市场的涌现 …… 300
12.4.2 新兴市场固定点 …… 301
12.5 创造性破坏网络 …… 304
12.5.1 创造性破坏的涌现 …… 305
12.5.2 平方根律固定点 …… 308
12.6 企业并购网络 …… 311
12.6.1 合并节点的 Java 方法 …… 311
12.6.2 合并加速创造性破坏 …… 312
练习 …… 313

第 13 章 生物学 …… 314

13.1 静态模型 …… 315
13.1.1 无标度属性 …… 315
13.1.2 小世界效应 …… 316
13.2 动态分析 …… 317
13.2.1 线性连续网络 …… 318
13.2.2 布尔网络 …… 320
13.3 蛋白质表达网络 …… 322
13.4 质量动力学建模 …… 324
13.4.1 质量动力学状态方程 …… 324
13.4.2 有界的质量动力学网络 … 327
练习 …… 328

参考文献 …… 330

第1章
Network Science: Theory and Applications

网络科学的起源

“新的网络科学”是一个简称为网络科学的新出现的研究领域，其实它相当古老，根源最早可以追溯到1736年。网络科学实质上就是将数学中的图论应用到各种领域的问题后，才在20世纪90年代后期以“新兴科学”的面目再次出现。然而图论自1736年孕育之初其实已被用于解决实际问题，当时瑞士数学家欧拉（Leonhard Euler）就使用图论解决了如何最优地环绕哥尼斯堡七桥的问题。

在随后的200年里，图论学家在这方面的研究却一直处于停滞状态。当然好的思想很难被长期埋没。到了20世纪50年代，匈牙利人、游牧数学家保罗·埃尔德什（Paul Erdos，1913—1996）发表了有关随机图的论文，重建了图论（并创建了称为离散数学的分支），使图论数学得以再次出现。埃尔德什精彩地将数学描述成一台可以用来将咖啡转换成定理的机器，他喜欢四处探访其他数学家，而不愿一直待在家中。如今我们将Erdos-Renyi（ER）随机图用做一种可与非随机图进行比较的基准。在1959～1960年间，用于构建随机图的ER生成程序成为第二个历史性的里程碑（参见表1-1）。

在20世纪60年代后期和70年代，图论被社会学家用于社会网络建模，用来研究群体内的人类行为。斯坦利·米尔格兰姆（Stanley Milgram）最早将小世界网络的概念引入到社会学研究中——从而激发了对网络拓扑如何影响人类行为以及人类行为如何影响拓扑的研究兴趣。“小世界难题”是贯穿这一时期的最活跃的研究主题。社会学家不禁会问：为什么人类能够通过非常少数的人就能将所有人相互联系起来，即使随着人口不断增长也是如此？

表1-1 重要事件的历史时间线

日 期	人 物	贡 献
1736	Euler	哥尼斯堡七桥问题
1925	G. Yule	偏好连接，Yule-Simon分布
1927	Kermack，McKendrick	第一个传染病模型
1951	Solomonoff，Rappaport	在随机网络中的传染扩散
1955	Simon	在词分析中观测到了幂律分布
1959	Gilbert	随机图的首次生成过程
1960	Erdos，Renyi	随机图
1967	Milgram	小世界实验
1969	Bass	人群中的创新扩散——非网络模型
1971	Fisher，Pry	产品替代品的扩散——非网络模型
1972	Bollobas	复杂图
1972	Bonacich	社会网络中的“影响”思想导致了影响图的出现
1973	Granovetter	求职网络形成了带有“弱链路”的聚类
1978	Pool，Kochen	小世界的首次理论检验
1984	Kuramoto	线性系统的同步

（续）

日 期	人 物	贡 献
1985	Bollobas	出版了有关“随机图”的图书
1988	Waxman	建立互联网的第一个图模型
1989	Bristor, Ryan	“购买网络” =网络科学应用于经济系统建模中
1990	Guare	创新词汇短语，“六度分隔” =百老汇戏剧中的剧名
1995	Molloy, Reed	具有任意度序列分布的网络生成
1996	Kretschmar, Morris	网络科学早期应用到传染病的扩散 =由最大连通的组件所驱动的传染
1998	Holland	在复杂自适应系统中引入涌现
1998	Watts, Strogatz, Faloutsos, Faloutsos	重新引发对最早的聚类例子“米尔格兰姆的小世界”的研究兴趣；小世界网络的第一个生成过程
1999	Faloutsos	在互联网中观测到了幂律分布
1999	Albert, Jeong, Barabasi	在 WWW 中观测到了幂律分布
1999	Dorogovtsev, Mendes	发现小世界属性
1999	Barabasi, Albert	构建无标度网络模型
1999	Dorogovtsev, Mendes, Samukhim, Krapivsky Redner	求无标度网络度序列的精确解
1999	Watts	对“小世界难题”的解释：高聚类、低路径长度
1999	Adamic	.edu 站点之间的距离显示为小世界
1999	Kleinberg, Kumar, Raghavan, Rajagopalan Tomkins	将 WWW 模型形式化为 Webgraph
1999	Walsh	在小世界网络中使用本地属性搜索困难
2000	Marchiori, Latora	谐波距离替代路径长度：为不连通的网络所做的工作
2000	Broder, Kumar, Maghoul, Raghavan, Rajagopalan Stata, Tomkins, Wiener	WWW 的完全 Webgraph 映射图
2000	Kleinberg	说明在小世界中使用“曼哈顿距离”的搜索为 $O(n)$
2000	Albert, Jeong, Barabasi	如果 hub 受到保护，那么无标度网络是有弹性的（互联网弱点）
2001	Yung	小世界理论应用分类：SNA、合作、互联网、商业、生命科学
2001	Pastor-Satorras, Vespignani	声称在无标度网络中没有传染病阈值；互联网容易受到 SIS 病毒的攻击
2001	Tadic, Adamic	使用局部信息可以加快在无标度网络上的搜索速度
2002	Levene, Fenner, Loizou, Wheeldon	导出 WWW 结构的增强的 Webgraph 模型不能只由偏好连接解释
2002	Kleinfeld	声称米尔格兰姆实验是没有根据的：小世界社会网络是一个“城市奇迹”
2002	Wang, Chen, Barahona, Pecora, Liu, Hong, Choi Kim, Jost, Joy	小世界中的同步等效于耦合系统中的稳定性
2003	Wang, Chakrabarti, Wang, Faloutsos	说明传染病的扩展由网络的谱半径、连接矩阵的最大特征值来决定
2003	Virtanen	完成了到 2003 年为止网络科学的成果调查
2003	Strogatz	蟋蟀鸣叫、心跳的同步
2005	NRC	网络科学定义
2006	Atay	具有度序列分布的网络中的同步——网络应用
2007	Gabbay	在影响网络中达成一致性——线性和非线性模型

斯坦利·米尔格兰姆著名的“六度分隔理论”实验说明，从美国全部人口中随机地选择两个人，他们之间的距离大约为 6 个中介点。在米尔格兰姆的实验中，要求堪萨斯州和内布拉斯加州的志愿者发送一封信给居住于马萨诸塞州波士顿和剑桥的不熟悉的目标。若不认识目标接收人，收件人会将信件转发给更接近目标的熟人。许多信丢失了，但是总会有些信件最终到达了目标地址，沿着中介链路的跳数范围为 2 ~ 10，平均跳数为 5.2。由此就诞生了小世界和“六度分隔”的概念。

在 20 世纪 90 年代后期，当很多其他领域中的科学家开始使用网络作为物理和生物现象的模型时，网络科学便已经进入了第三个阶段，目前正朝着成为一门独立的学科发展。特别是，Duncan Watts、Steven Strogatz 和 Albert-Laszlo Barabasi 的先驱工作——将网络科学应用到物理领域中，再次激发了对网络进行数学分析的兴趣。Watts 将小直径非常稀疏的网络（小世界）与各类现象如材料中的相变、生物有机体的功能以及电网的行为等同联系起来。那么简单的图模型是如何解释现实世界行为的多样性呢?

Strogatz 研究了网络结构对物理学中复杂自适应系统的影响，也解释了为什么哺乳动物心跳会规则同步，以及为什么某种萤火虫会在没有集中控制的情况下有节奏地共鸣。似乎是生物有机体倾向于在没有全局知识的情况下同步其行为。在本书中，我们将深入理解物理学和生物学中如何以及为什么会发生网络同步，也解释了一组人在什么条件下会达成一致性，如何才能做好产品营销策划，以及企业是如何实现垄断的。同步是“生物网络”结构附带产生的结果。

Barabasi 和他的学生们开辟了另外一条研究路线，发明了*无标度网络*——带有 hub 的非随机网络。在很多互联网和 WWW 结构的研究中，Barabasi 等人发现了分散的互联网的一种涌现属性——在没有集中规划时，这种属性出现在由少数非常受欢迎的称为 hub 的站点和大量“不受欢迎”的具有很少链路的站点组成的结构之中。互联网拓扑不再像一个 ER（Erdos-Renyi）网络那样是随机的，而是非常不随机的。实际上，具有 k 条链路的站点概率服从幂律分布，对于大的 k 值来讲概率会很快减少。进一步来讲，他们推测这是称为*偏好连接*的微观规则导致的结果，即站点获得一条新的链路的概率直接与它已有的链路数成正比。如此一来，站点具有越多的链路，那么它就会获得更多的链路——这也就是所谓的“富者越富”的现象。

无标度网络是非常不随机的。这种发现为各个学科（无论是政治学还是社会学、生物学和物理学）丰富的论文发表创造了条件。为什么有这么多自然现象服从幂律分布而非正态分布或指数分布呢？这些再次为深入研究组织、生物和物理的结构以便解释由幂律分布引出的问题搭好了舞台。

网络科学的当前状态最好描述成正处于“发展阶段”。到达现代状态，大约也只经历了十余年的历史。每个月都会有一些新的发现，这意味着本书可能很快就会过时。因此，作者力图集中于基础理论研究，希望其成果能经得起数十年之久的考验，而非钻到虽然有趣但会分散注意力的研究中去。

本章的目的在于定义这门新出现的学科，即所谓的*网络科学*，并列举出使它处于目前状态的历史年表上的关键事件。我们调查了过去 270 年中导致当前状态的技术，以松散的网络“规则”集合作为结论。我们将详细研究如下方面：

1. 网络科学可以以多种方式定义。我们大致地可将它定义成对网络结构/动态行为的理论基础研究及其在很多子领域的应用。网络科学既是理论又是应用。

2. 网络科学的历史。

3. 网络科学的核心概念或原则（至少）是结构化的、具有动态特性、自底向上演化、自治、拓扑、有用性（power）、稳定性和涌现性。本章会详细地解释每一个概念。

4. 网络科学将网络同步的涌现与线性耦合系统的稳定性联系起来，我们对此给出了新的视角。这种视角集成了很多应用底层的概念，像传染病传播模型、各种形式网络涌现的动态性及在分散的领域（如生物、物理和营销）中观察到的行为。我们断定这些应用的底层行为只不过是更一般的线性系统稳定性的特殊情况而已，也就是说，传染病的扩散、群体内部达成的一致性、电网的稳定性等都是耦合线性系统分析的应用。所有这些看似分散的行为可以使用谱分析工具进行解释和分析。

1.1 什么是网络科学

“未来军队网络科学应用委员会”（The Committee on Network Science for Future Army Applications）以多种方式、不同的详细层次定义了网络科学（National Research Council（NRC，美国国家研究委员会），2005）。或许 NRC 给出了最简单和最直接的定义：“基于科学方法研究的有组织的网络知识。”这种定义意味着将网络科学与各种使用它的技术区别开来，例如，将底层所隐含的网络科学与如互联网等的具体技术相区别开来。

当然现在网络科学还不够成熟，还不能将它与底层技术根源彻底相分离。委员会发现在使用网络科学的每一个子领域中都有着各自不同的工作定义。通信工程师将网络看做由路由器和交换机组成的系统；社会学家将网络看做表示人与人之间社会交往的影响图；市场营销人员将网络看做购买者人群的集合；物理学家将网络看做是相变和磁场的模型等。生物学家使用网络比喻来理解传染病、基因和细胞内的新陈代谢系统，而电力工程师谈到网络则会联想到电网。网络科学在各领域的人们眼中，有着不同的命名、词汇和分析方法。

或许与网络科学的定义相比，更容易定义什么是网络。从这方面来讲，网络的概念更加普遍，即使术语本身并非如此。委员会是“通过结构（例如节点和链路）及其行为（网络“所做”是节点和链路之间相互作用的结果）”描述网络的。进一步来讲，“网络总是可观察事实（但不是事实本身）的一种表示或模型。”网络是表示某种真实情况的图。

上述实用性的定义标示出了网络科学的两个关键组成：（1）它是研究表示某种实际的节点和链路集合的结构；（2）它是研究汇聚节点和链路的动态行为特性的。那么人们会问：“随着时间的推移，网络演变时会发生什么呢，以及为什么会这样发生？”网络科学最重要的结果似乎是将形式与功能以及结构与行为关联起来。当前，最有趣的行为发生在物理、生物和社会系统中。节点可能是人、分子、基因、路由器、变压器（电力网）、Web 页面或研究出版物。链路可能是指朋友之间的友谊、接触传染、神经键、缆线、互联网链路或著作目录的引用。从这一点来讲，网络科学是实际的抽象但非实际本身。但是如果抽象可以解释实际系统的行为，那么网络科学不仅极其有趣而且也非常有用。

网络的结构部分很容易通过图论来建模。具体来讲，网络本身可以用集合的形式定义成 $G = \{N, L, f\}$，其中 N 为节点的集合，L 为链路的集合，$f: N \times N$ 为定义结构 G 的映射函数——节点是如何通过链路连接起来的。映射函数包含足够的信息，可以使用点作为节点、线条作为链路在平面纸上作图。但是集合 G 不足以定义网络的动态行为。

网络的动态部分可以通过一组用来控制节点和链路行为的*微观规则*来定义。这些规则在*微观层次*给定，从而将它们与网络的*宏观层次*行为区别开来。详细来讲，微观层次规则决定着链路和节点的行为，宏观层次规则决定着网络全局特性的涌现。例如，*偏好连接*（链路会受到具有很多链路的节点的吸引）已经是一个微观规则，而描述网络*度序列*分布的幂律分布为宏观层次规则。作为网络科学家，我们主要考虑通过研究微观规则理解宏观层次属性——有时也要反过来进行。

网络 G 的完整定义必须既要包括结构的信息也要包括行为的信息。例如，$G(t) = \{N(t),$

$L(t),f(t)\}$ 是动态维数的网络 G 的集合论定义——$G(t)$ 是一个时间 t 以及随时间变化的节点和链路数、值及映射的函数。$G(t)$ 的实际行为用算法来表示，一般以计算机算法的形式来表示。在本书中，我们使用 Java 程序设计语言表示微观规则。将上述这些定义放在一起，带有结构和行为元素的简洁网络定义表示如下。

网络定义

$$G(t) = \{N(t),L(t),f(t):J(t)\}$$

其中，t = 仿真或实际的时间；N = 节点，又称为顶点或“个体”（actor）；L = 链路，又称为边；$f:N \times N$ = 连接节点对以产生拓扑的映射函数；J = 描述节点和链路行为随着时间变化的算法。

在研究网络时，我们给出一个严格的包括网络的结构和行为的网络科学定义：

网络科学定义　网络科学或网络的科学，是研究网络结构/动态行为并将网络应用到很多子领域的理论基础。当前已知的子领域包括社会网络分析（SNA）、协作网络（书目引用、产品营销、在线社会网络）、人造的涌现系统（电力网，互联网）、物理科学系统（相变、浸透理论、Ising 理论）和生命科学系统（传染病、新陈代谢过程、遗传学）[⊖]。

从该定义应该很清楚地了解到网络科学实质上就是系统科学。此外，因为网络经常为复杂系统建模，它与古老的复杂自适应系统领域密切相关。实际上，网络科学集成了来自复杂自适应系统（涌现）、混沌理论（同步）和平均场理论（物理学）的思想。网络科学是所有这些学科的交叉学科，它将相邻学科的思想拉到了一起。

1.2　网络科学简史

网络科学已经存在很长时间了，特别是将图论看做它的起源时（参见表 1-1）就更是如此。但是由于网络科学的动态特性以及在很多其他学科中的应用，它并不仅局限于图论。一般来讲，网络科学根植于图论、社会网络分析、控制论，尤其最近的物理学和生物科学。从某种意义上来讲，网络科学就是许多领域汇聚的结果。

从长远来看，网络科学似乎经历了至少两次重大的变迁：从数学理论到图的应用，从应用到“相互连接到一起的所有东西”的一般集合。相应地，我们将网络科学的历史划分成三个主要阶段：（1）早期的网前阶段（1736—1966），当时网络科学还是数学中的图；（2）中期网络阶段（1967—1998），当时网络科学还没有称为“新的网络科学”，但实际上网络应用已经在研究文献中出现了；（3）现代阶段（1998—），网络科学的先驱们为网络科学的当前定义打下了基础，并证明这些基础具有现实意义。在现代阶段，当网络科学应用到各个似乎互不相干的领域时，倡导者们开始论述网络科学的普遍性。

1.2.1　网前阶段（1736—1966）

网络科学第一个已知的应用是欧拉对哥尼斯堡七桥问题（Euler，1736）的处理。这一问题非常重要，因为它建立了图论，并证明了对实际情况进行的抽象的确有助于解决实际问题。欧拉通过抽象的数学对象——顶点（节点）和边（链路），轻而易举地定义了物理系统的静态结构。在抽象层次上进行逻辑推理，他证明了哥尼斯堡市民不可能游行通过城市并返回而不穿越 7 桥中任意一座两次。

哥尼斯堡七桥问题　哥尼斯堡是一座城市，有一部分陆地在普雷格（Preger）河中央的一座小岛上，其他部分被河汊所隔开。通过七座桥，市民能从四块陆地中的任意一块到达另外一块。

⊖ Ernst Ising（1900—1998）提出了以他的名字命名的从顺磁性状态到铁磁性状态的相变模型。在某点处，足够多的原子以同一方向排列造成了铁中的磁性。这是一种从显著的随机极性（原子互消）状态向最小能量状态或显著的对齐原子的状态迁移的结果，因此便产生了全部的磁效应。

四座桥将河的两岸与小岛连接起来，两座桥穿越河的分叉，一座桥将小岛和位于分叉之间的陆地连接起来。城市的长老召见欧拉，咨询是否可以游行穿越整座城的各块陆地并且仅通过每座桥一次。对该问题的解答我们将在第2章中给出。

欧拉是一个伟大的数学家，瑞士当局在他死后48年才公布了他的全部论著。如今，他仍旧被尊称为图论之父，其遗作成为网络科学结构化部分的理论基础。

自270年前欧拉奠定了基础之后，数学家发表了数以千计的图论成果。图论在计算机科学和电子工程以及很多其他应用学科中特别有用。但是就网络科学而言，下一个主要的发展是在1925年，Yule首次观察到进化中的*偏好连接*（Yule，1925）。Yule的工作似乎与网络科学无关，但是他的理论思想在20世纪90年代解释互联网和WWW（万维网）的演变时得以重现。偏好连接解释了为什么无标度网络会存在于自然界和人造系统中。

偏好连接是一种在很多学科中观测到的简单涌现行为。在网络情形下，据称网络会随着节点和链路的添加而增长，但不是随机而是有偏好地进行的。假设网络是从三个节点并在三个节点中的一个节点对之间添加一条链路开始的。现在假设新节点的添加是以固定的时间间隔进行的，也就是，让网络以每次将一个节点连接到现有的节点上的系统过程增长。新节点应该如何连接到现有的节点上？随机连接是我们可以使用的算法之一：随机地选择现有的节点，并通过在节点对之间添加一条新的链路将它与新节点相连接。

另外一种可选（和达尔文主义的）算法如下。以与现有节点链接数成正比的概率将新节点连接到已有的节点上。节点上的链路数称为*度*。这种连接规则也就偏好优先选择度高的节点。如此一来，新的节点更有可能连接到包含有两条链路的节点上而非只有一条链路的节点上。新的节点偏好连接到度更高的节点上。

偏好连接描述一种涌现过程——这种过程导致的网络拓扑不能通过检查局部算法或*微规则*就能弄明白，重复应用偏好连接的结果不会明显地导致具有*服从幂律分布的度序列分布*的网络。直到70年后，当A.-L. Barabasi和R. Albert说明如何通过重复应用偏好连接创建无标度网络时，才认识到这一点。

在1927年，当Kermack和McKendrick发表了生物群中传染病传播的第一个数学模型（Kermack，1927）时，出现了另外一个似乎无关的发现。Kermack-McKendrick *传染病模型*是一个非网络模型，但是它为两个重要的创新做好了准备：（1）它解释了传染病是沿着连接（单独的）节点的（社会）链路扩散；（2）它碰巧描述了*新产品的接受*（即*技术的扩散*）以及产品信息如何像传染病一样在社会网络中扩散开来。第一次创新非常重要，因为Kermack-McKendrick传染病模型推导出了病毒在网络（如互联网）中的传播规律。事实上，我们证明了扩散速率以及传播的持久性完全由网络的拓扑结构和传染病的传染性来决定。因此，某些网络要比其他网络更容易受到传染病的影响。进一步来讲，理解网络拓扑和传染病的传播之间的关系，可以告诉我们如何才能更好地*阻止*它的传播。

Kermack-McKendrick模型的第二个重要意义是应用到商业领域中的新产品营销上。在社会网络中信息的扩散（广告或散播消息）很像传染病的传播。网络的什么属性加速或延迟了这种病毒式的传播？在这种情况下，商人想要知道如何提高传染性。

Solomonoff和Rappaport最早将传染病的思想应用到网络上（Solomonoff，1951）。因此这样的联系在50年前（即大约20世纪50年代中期）就存在了，但是网络拓扑和传染性之间的联系需要等待最近的研究突破。Solomonoff和Rappaport假定网络是随机的。如今我们知道随机网络很少存在于现实世界中——在应用到社会网络中时，随机性与实际情况相差很远。进一步来讲，我们现在知道非随机网络的结构可以对其功能有极大的影响。

事实上，作为在行为及偏好连接与非随机分布之间联系的一个因素的非随机性思想是Simon

于 1955 年想到的（Simon，1955）。Simon 清楚了解 Yule 在偏好连接上的工作以及植物种属之间的品种分布。Simon 的观察证实了在自然界和人工系统中的幂律分布的合理性，也就是说，词在文档中出现的频率、科学家发表论文数量的分布、人类城市的分布、财富的分布以及生物品种在种属中的分布，所有这些现象都满足幂律分布（Mitzenmacher，2004）。各种证据显示，在人们观察到网络中非随机性很久以前，实际现象中的非随机性就已经存在了。但是系统中的非随机性和图论之间的联系还有待进一步探索。

到了 20 世纪中期，科学推理说明自然界可以建模成一个随机过程，并因此做成一张随机图。那么随机图的什么属性使其成为良好的模型呢？Gilbert 演示了如何构建随机图，首先构造一张完全图，然后删除随机选择的链路直到需要数量的链路为止（Gilbert，1959）。但是他的笨重的算法很快就被 Erdos 和 Renyi 的优美的、广受推荐的算法所超越（Erdos，1960）。Erdos-Renyi（ER）算法因其简洁性当前仍在使用。带有 n 个节点的网络是通过在随机选择的节点对之间插入链路来构造的。重复该过程直到插入了 m 条链路为止。

尽管 ER 算法并不完美——因为它可能会留下某些孤立节点，而且若不对它略加修改也可能会在网络中插入重复和循环的链路，但是它已经成为计算机生成随机网络的标准方法。Gilbert 和 ER 算法是第一个创建网络的生成方法。自此以后提出了更多的方法，实际上在本书中我们也给出了更多的建议。如今的计算机算法可以生成任意指定的拓扑、具有任意数量的节点和链路的网络，参见第 7 章。

到了 20 世纪 60 年代末，网络科学还不存在，但是其种子已经植入不连贯的、似乎无关的学科中了。当然这还需再花费数十年时间，才能将在迥然不同的学科中分散的研究汇聚成目前我们所谓的网络的科学。

1.2.2　中期网络阶段（1967—1998）

在 1967 年曾进行过一个惊人的实验，推动了网络科学从纯图论进入科学研究。斯坦利·米尔格兰姆著名的“六度分隔理论”实验似乎非常单纯，但是回顾一下，它标志着一个转折点。斯坦利·米尔格兰姆邀请了来自堪萨斯州和内布拉斯加州的实验者参加“通信项目”以便“更好地理解美国社会的社会联系”。实验要求他们发送一个文件夹，横跨美国到达实验指定的目标人。具体按照以下步骤进行（Yung，2001）：

1. 将你的名字添加到该信纸的名册末尾，以便于接收到这封信的人知道信是发自谁的。

2. 拆掉一份明信片，填写好，将它发回哈佛大学。不需要额外张贴邮票。这样做便于我们在文件夹靠近目标人的过程中进行跟踪。

3. 如果某个人知道目标人，就直接将该文件夹发送给他/她。当然这仅在你以前遇到过目标人并且相互知道姓名的前提下才这样做。

4. 如果某个人不知道目标人，就不要试图直接跟他/她联系。而是将该文件夹（明信片及所有）邮寄给很可能比你更熟悉目标人的熟人。你可以将文件夹发送给朋友、亲戚或熟人，但是必须在知道其姓名的基础上再这样做。

给实验者 24 小时转发他们的文件夹，并会收到一封感谢他们所作努力的证书。大多数文件夹没有到达目的地。但是到达目标的文件夹以比意料之中少得多的步骤到达。几百万人中，文件夹在到达目的地前仅经过少数几个中介。这种通过大量人所精选出来的路径，就是小世界效应的证据，米尔格兰姆进行实验的社会网络就是目前所谓的小世界网络。中介数平均为 5.2。

一个陌生人怎么可能与另外一个陌生人在平均少于 6 步的情况下就能联系起来呢？米尔格兰姆不得不得出如下结论：社会网络组织形成了一个非随机的网络。不是随机地从一个人到另外一个人，而是文件夹采用了捷径到达目的地——所沿着的路径不是提前规划好的，也不能保证在发送者和接收者之间一定存在一条中间链。

假定每个人平均认识500个其他的人，平均到达目的地的比率约为二十万分之一，如果成功到达的文件夹所采取的路径真的是随机的话，中介数应该比6跳大很多倍。但是，从发件人到接收人成功到达的文件夹平均以5.2步（跳）到达目的地。在图论中传输的距离为5或6跳，因为5或6个人中转了文件夹。

米尔格兰姆的实验激发了由John Guare创作的百老汇的戏剧和电影，名为《Six Degrees of Separation：A Play》（六度分隔）（Guare，1990）。术语一旦扎根，小世界社会网络的思想便会开花结果，便导致了有目的地创建许多其他社会网络。由弗吉尼亚大学Brett Tjaden创建的《Kevin Bacon游戏》就是这种例子之一[⊖]。这种网络将在同一部电影中出现过的演员联系在一起。Kevin Bacon和其他演员之间的距离等于从代表Kevin Bacon的节点到代表其他演员的节点的跳数。

米尔格兰姆得出结论，人类社会比“真实世界”要小得多，因为仅用6跳就能将一对陌生人连接起来，而且与他们居住在哪里无关。他将此称为小世界网络问题。几十年后Watts和Strogatz再次对小世界网络感兴趣，并将它介绍给物理学家和生物学家。他们对大的稀疏网络的技术分析严格地将小世界定义成在随机选择节点对之间具有相对短距离（跳）的网络，即使网络规模扩大也是如此。详细地讲，当网络规模大小以$O(n)$增长时，网络直径以$\ln(n)$增加，这里n为节点数。

小世界思想与Granovetter的“弱联系”理论相关，推测社会网络既包含将社会捆绑在一起的强联系（直接连接）也包含弱联系（远距离连接）（Granovetter，1973）。在一篇才华横溢的论文“弱联系的力量”中，Granovetter提出社会网络既要由朋友和家庭成员之间的强连接，也要由偶然认识的人的较弱的、远距离连接所决定。这就可以解释为什么能以少数几跳就能跨越一个很大的稀疏网络。社会网络的链路就像高速公路和街道一样，高速公路具有少数交界点（节点）使你能够旅行很远距离而不用停下来。另一方面城市街道使你能够在密集的邻居中精确地找到某个人或房屋。高速公路（弱联系）使你可以快速到达某一区域，而街道（强联系）会定位特定的人或房间。

White指出了米尔格兰姆的实验偏差并给出了修改建议，结果为平均7个节点数（White，1970）。Hunter和Shotland将实验建模成Markov过程以便确定组之间的平均距离（Hunter，1974）。Pool和Kochen于1978年给出了最早已知的小世界网络的理论分析（Pool，1978）。受米尔格兰姆的实验激发而发表的论文在1978年后急剧下降，又经过20年后才重新得以恢复（Kleinfeld，2002）。

同一时期，先驱们的工作在很快就要在关联起来的学科中展开：Bass、Fisher和Pry对新产品的接受按照传染病的传播方式进行建模（Bass，1969，2001；Norton，1987）。这种工作将Kermack-McKendrick传染病模型扩展到新的营销领域，并为基于网络的产品扩散模型提供了方法。这些模型已经证明是商业营销的一个强大工具。但是当应用到社会网络时传染病模型会起作用吗？我们解释了对于单个产品和随机网络技术扩散（新产品的采用）服从Bass和Fisher-Pry方程。我们也阐述了从随机人群中产生的垄断是由于偏好连接引起的。但是我们在为多产品网络中的产品竞争建模时，发现Bass/Fisher-Pry模型具有局限性。这些结论暂时还没有在其他地方报道过。

在社会科学中，图论被用来解释很多其他社会互动现象。Bonacich或许是第一个认识到社会网络中的影响可以使用网络连接矩阵进行数学表示的社会科学家（Bonacich，1972）。使用节点代表个人，在有向链路上的权重[⊜]表示一个个体对其他个体的影响度。如果A影响B的决策，B又

⊖ 该游戏可以在http：//www.cs.virginia.edu/oracle/上找到。

⊜ 权重为小数：$0 \leqslant$ 权重 $\leqslant 1$。

会影响 C 的决策等，对组达成一致性的全部影响是什么？影响链将会沿着网络传播并且最终稳定在一个共识吗？Bonacich 断定终究会达成一致性，并且提议一致性可以通过将权重连接矩阵提高到 n 次幂来计算获得，这里 n 为网络中的节点数。正如结果证明，社会网络中影响的扩散并不像 Bonacich 模型建议的那么简单，但是 Bonacich 发起了一项一直持续到今天的研究。

市场营销专家注意到高连通的人是*超级传播者*（即加速 buzz 扩散的人），新产品信息扩散就是因为他们是高连通的缘故（Rosen，2000）。但是社会科学家很久以前就知道中间人或中介（即连接其他个体的个体）的有用性。如果个体 A 仅通过个体 B 才能够和个体 C 联系，那么 B 就有超过 A 和 C 的有用性。如此，社会科学家将介数定义成必须通过一个个体与其他个体联系的路径数。因此，除了连通性外，个体通过充当中介而导出了影响。介数要比连通性能给予个体更多的影响吗？这是我们研究影响网络时所提到的问题。参见第 10 章。

影响扩散——不管它是产品营销、疾病传播还是群组内达成一致性，它都是一种信号传播。信号沿着链路传播，并且以某种方式影响节点。例如，节点的值可能是邻接节点的平均值。在一个*基尔霍夫*（Kirchhoff）网络中，节点值等于输入总和值与输出总和值之差。不管为节点分配值的本地微观规则如何，信号流经网络的概念看上去是流行病学、同步、影响和组内达成一致性的共同机制。

更严格地讲，网络可被看成是一种耦合系统。系统是由节点（所取值称为*状态*）和链路（建立输入和输出到节点）构成。网络的状态是所有其节点状态的联合。信号（值）沿着链路传播，从一个节点到另外一个节点，并且更改节点的状态。如果我们画出状态随时间的变化图，我们可能会观察到振荡、抑制或收敛到某一状态（即所谓的*固定点*），并且永远地停留在那里。在什么条件下网络振荡或收敛呢？这是一个一般性的问题，我们将会在第 10 和 12 章中解答。

我们将证明传染病的传播、生物系统的同步、社会网络中的达成一致性、新产品的扩散都是网络同步的不同形式。当网络的节点值到达某一*固定点*时，即一旦到达某值就会停止变化，那么网络被说成是同步。我们解答了这一问题："为了网络同步，充分和必要的属性或条件是什么？"答案推导出了网络中稳定性的一般理论。

Kuramoto 为研究耦合线性系统中的同步提供了数学基础（Kuramoto，1984）。他的工作在十年以后影响了 Strogatz，并且对网络科学和控制论间的结合有着重大影响。例如，Kuramoto 的工作引导 Strogatz 观察小世界网络中的自动同步。Strogatz 声称同步是所有小世界的属性（Strogatz，2003）。这后来被证明是错误的，但是他激发了对自动同步的各种生物系统的网络模型的进一步研究。现在我们知道，网络中必须存在其他条件才能使它们同步。

到了 1998 年，网络科学的基础已经建立起来了，但是将原理应用到实际的兴趣还没有开始激增。起始于 20 世纪 90 年代早期的快速增长的互联网，为致力于规范人造的但是高度分散的互联网现象的新一代研究人员提供了动力。Waxman 于 1988 年提出了互联网的静态图论模型（Waxman，1988）。我们将这种网络称为 Webgraphs，因为这种网络使用图论来理解万维网。还需要大约十年的时间研究人员才将图论与动态增长的互联网和万维网联系起来。但是一旦做到了，网络科学的时代就到来了。

1.2.3　现代阶段（1998—）

网络具有静态属性和动态属性。利用静态图属性，例如直径、平均路径长度、连接矩阵和聚类系数，可以为网络提供一种分类的方法。例如网络的*度序列分布*就是具有度为 d 的节点的百分比与 d 之比的直方图。随机网络具有服从二项式分布的度序列分布，无标度网络的度序列服从幂律分布。小世界网络具有相对较小的直径和平均路径长度，无标度网络具有 hub。这些是根据其结构来分类的各种方法。但是仅根据静态结构分类是不够的。

网络也具有动态属性，比如它们同步时的固定点，以及当偏好连接可操作时将节点有偏向

地链接到一起。动态网络是变化的。从某种初始状态开始，动态网络可以迁移到第二种、第三种、第四种和更高的状态，直到要么继续循环要么到达最终状态——它的固定点为止。从最初状态到某种将来状态的演变就是一种涌现形式。因此，到达固定点的网络永远不同于振荡的网络。这样，我们就可以按照动态属性对网络分类了。

网络科学既研究网络的静态属性又研究其动态属性。在本书中，我们重点在于将涌现作为一种理解和描述网络科学动态部分的一种方法。我们进一步将涌现方法分成两部分：更改网络拓扑的部分（例如偏好连接）和更改网络状态的部分（如同步）。我们将说明任意结构的网络可以作为涌现过程的固定点来产生，由此重联链路直到出现理想的拓扑为止。在最后一章，我们证明稳定状态是从具有某种初始状态和某种静态属性的网络的混沌中涌现的。

Holland 用更加通俗的成语如“积少成多”等定义涌现（Holland, 1998）。这是对网络演变时所发生事情的精确描述。一系列微观层次的更改会随着时间而积累，直到实现了网络中的宏观层次更改。涌现意味着全局属性的重要更改来自许多局部层次的小的更改。涌现在现代网络科学释义中扮演着重要的角色。从某种意义上来讲，它解答了用什么来定义研究领域的难题。

Watts 和 Strogatz 通过说明小世界模型的通用性以及实用性重新引发了人们对小世界网络的兴趣（Watts, 1998; 1999a; 1999b）。他们提出了一种简单的涌现过程（称为*生成过程*），用于构造小世界网络。设想虽然很简单，但却很精彩：最初用规则结构构造一张图；接下来，以概率 p 重联每条链路，以便于 $p * m$ 条链路是随机的（重定向到随机选择的节点上）。参数 p 为重联概率，而 m 为原来规则图中的链路数。

Watts-Strogatz 生成过程是可调整的——提高 p 也会提高小世界的随机性。这就意味着我们可以生成具有任意随机性层次（熵）的网络，如果你愿意，可以通过调整重联概率 p 实现——低重联概率产生一个非随机结构，而高概率产生一个随机结构。该算法是第一种涌现的例子——网络的结构演变而非状态演变。

Watts-Strogatz 网络介于随机网络和非随机网络之间。在某一重联概率（被称为*转换点*）处，网络会从大多数结构化的网络迁移到大多数随机的网络。一般来讲，转换点非常小——$p^* = 2\% \sim 3\%$。Watts 和 Strogatz 为转换附加了含义，假定它对应于材料中的相变（从液态变成固态，或磁化等）。如此一来，转换点又称为*相变阈值*。

Watts 和 Strogatz 说明如何生成一个任意小世界网络后，小世界已经不再局限于如米尔格兰姆实验中的社会网络了。不仅如此，在自然界和人工系统中也都观察到了小世界网络，这就意味着它们是某种通用的模型。电影演员数据库、美国西部电力网和秀丽小杆线虫（C. elegans）的神经网络具有共同之处吗？答案为：它们都是小世界的网络。这就不仅是偶然现象了。

小世界不再是似乎唯一可能普遍存在的网络分类。1999 年有了大量的发现：M. Faloutsos、P. Faloutsos 和 C. Faloutsos 在他们的互联网图模型中观察到幂律分布，Albert、Jeong 和 Barabasi 在对 WWW 的研究中也获得了相似的结论（Faloutsos, 1999; Albert, 1999）。小世界具有短路径长度的性质，但是幂律分布网络具有 hub——度非常高的节点。Barabasi 及其学生发现许多人造和自然网络的度序列分布服从幂律分布。服从幂律分布的网络意味着具有 hub，并且很多其他节点具有比平均值少得多的链路。这种对于 hub 的偏爱不平衡似乎与自然界的正态分布相反。问题在于研究人员忽略了太多的所发现的按照幂律分布结构化的系统。

Barabasi 和 Albert 总结了带有 hub 的非随机网络的概念，并且为生成*无标度网络*提供生成过程（Barabasi, 1999）。定义来自于如下发现：如果 $f(ax) = a'f(x)$，那么函数 $f(x)$ *缩放*，这是幂律分布所造成的。因此如果度序列分布不服从幂律分布，$h(x) = x^{-q}$，那么就很显然 $h(ax) = (ax)^{-q} = (a^{-q})h(x) = a'h(x)$。但是最重要的贡献不在于定义，而是观察到的结果：无标度网络节点频率与度 d 之间的关系曲线会随着 d 的增加而急剧下降。

1999 年到 2002 年间，认识到了小世界和无标度网络的重要性之后，在数学家、物理学家和社会科学家之间就造成了所谓的吞食鱼效应。Dorogovtsev、Mendes、Samukhim、Krapivsky 和 Redner 导出了纯无标度网络的幂律分布的精确描述公式，并且说明它能用来描述很多生物系统（Dorogovtsev，2000；2002a；2002b；2003）。Kleinberg、Kumar、Raghavan、Rajagopalan 和 Tomkins 建议用术语 Webgraph 描述 WWW 的网络模型（Kleinberg，1999）。Broder、Kumar、Maghoul、Raghavan、Rajagopalan、Stata、Tomkins 和 Wiener 首次完全将 WWW 映射成 Webgraph，并揭示了它的结构，它不是随机的。Kleinberg 根据源和目的地之间的“曼哈顿距离”为米尔格兰姆的实验给出了一个正式解释。“曼哈顿距离”被定义为从纽约曼哈顿为源到目的地道路交叉口所经过的街区数目。Kleinberg 证明了仅经过 $O(n)$ 步就能导航通过这种小网络（Kleinberg，2002a）。

到了这种激动人心发现的最后，网络科学的基础地位就稳固了下来。但是这些模型是做什么用的呢？如果无标度和小世界拓扑正如数学家和物理学家所声称的那样，那么在自然界和人工系统中就应该容易找到实例。进一步来讲，如果无标度和小世界拓扑具有深奥的含义，我们应该能够从理论中导出普遍的真理。事实也的确如此。

Albert、Jeong 和 Barabasi 观察到无标度网络在受到随机攻击时会非常富有弹性，但是在遭到对 hub 的系统攻击时就非常脆弱（Albert，2000）。这具有某种实际意义——随机攻击最有可能打击并摧毁仅具有少数链路的节点，因为无标度网络具有很多这种节点。相反，既然 hub 很少，hub 不大可能受到攻击。但是 hub 具有很多链路，因此它的消亡会破坏很大部分的网络。假设 p_r 为拆除破坏网络的受损节点的百分数。当 p_c 很高时，网络是有弹性的，因为拆除破坏网络必须去掉许多节点。当 p_c 很低时，网络是很脆弱的。在仿真中，Albert 等人发现随机网络的阈值 $p_c = 28\%$，在随机攻击下无标度网络接近 99%。也就是说，当其节点平均有 28% 被破坏后，随机网络被拆除破坏。但是当 hub 被系统地攻击时情况就会完全转变——仅攻击 18% 的节点就能拆除破坏一个无标度网络。如此一来，当它的 hub 被定为目标时，无标度网络要比随机网络更脆弱。

Albert 等人在实验中所提出的问题，在本书中我们做了解答：网络中弹性（风险）的含义是什么？我们扩展了 Albert 等人的结论，并根据节点的价值、它们的度以及风险公式 $R = T * V * C$（这里 R 是风险，T 是威胁概率，V 是脆弱性概率，C 是后果或毁坏），为任意网络分配一个风险属性。我们说明了：如果根据由 Al-Mannai 和 Lewis 提出的算法（Al-Mannai，2007）将目标强化资源布置到节点和链路上，任何网络可以进行优化保护而不被拆除破坏。Al-Mannai – Lewis 算法首先保护高价值 hub，最后才是保护低价值节点。

在相关的工作中，Pastor-Satorras 和 Vespignani 观测到：人群构成一个无标度网络，这个网络没有能阻止传染病重发的最小传染阈值（Pastor-Satorras，2001）。一旦传染病进入一个网络，它就反复起、落。持久的传染是实际情况——它们既发生在人际网络上，也会发生在互联网上。如果互联网是一个无标度网络，那么如何才能阻止病毒在互联网上永久传播呢？

Wang 和合作者证明了 Pastor-Satorrus 的最初论断在一般情况下是错误的（Wang，2003a）。作为替代，传染的持久性不是由网络的度序列来确定的，而是由它的谱半径所确定的，即定义为网络的连接矩阵的最大非平凡特征值。因此，网络拓扑决定了它是否易于受到感染，但是却不是因为它是无标度的。这种结论无论是对互联网还是对人类的病毒对抗都具有很深远的意义。这对从事产品市场营销人员来说也有某种启发。

系统网络模型对于理解弹性、风险、传染病和群组中人员的社会交往都有深刻的影响。它对于理解耦合线性系统的混沌行为也被证明是创新性的。Wang、Chen、Barahona、Pecora、Liu、Hong、Choi、Jost、Joy 等人将线性系统分析应用到任意网络上，并且说明任意网络的稳定性是其拓扑的函数（Wang，2002a，2002b，2002c；Barahona，2002；Liu，2002，2003，2004a，2004b；Hong，2002；Jost，2002）。我们将这些结果扩展到几种网络类型：Atay 网络，一种由 Atay 研究的

动态网络，使用局部平均算法计算节点的状态（Atay，2006）；以及一种新的网络，称为基尔霍夫网络，通过满足基尔霍夫第一定律试图稳定它的节点值。

在动态网络中的固定节点解（同步）的涌现已经成为理解很多自然现象的强大而通用的工具。Strogatz 认为人的心脏跳动、蟋蟀的鸣叫以及其他生物系统之所以自然同步，是因为它们是小世界的（Strogatz，2003）。但是我们可以证明网络同步与小世界拓扑没有关系，一切都是与连接矩阵的拉普拉斯变换以及网络中的回路长度有关。我们检查 Atay 网络的稳定性，它在市场营销中的应用，以及对蟋蟀鸣叫的理解，并且阐明了同步是包含三角形子图的涌现网络属性。结果证明，小世界网络具有充足的三角子图！不仅如此，网络可以通过在其中某一个节点上添加一个三角子图来实现同步。

基尔霍夫网络图会更加复杂——定义成有向流网络，这里每一个节点的状态就是进入链路值的总和减去出链路值的总和。基尔霍夫网络是电网、电子机械伺服系统、航空线路等的抽象。因此，它们最好不出故障，这一点很重要——但是实际情况并非如此，即使断开或突然更改一个或多个节点值后，它也会很快稳定下来。奇怪的是，当网络中的回路长度为相对素数时，基尔霍夫网络在显示出混乱的混沌行为后会突然同步。这对于设计自稳定电网、自纠正传输系统或许还有生物系统的自愈都有着重要的应用。

也可以在社会网络中观察到导致同步或不同步的相似的稳定性问题。假定一组人员尝试达成一致性（Gabbay，2007），组中的每个成员从初始位置开始，并通过对最近的相邻者施加正面的或负面的影响，试图影响其相邻者的位置。影响的传播像传染病一样，但是不论传染与否，影响都会改变邻接节点的位置。例如社会网络中的每一个节点都可能被调整到其邻接节点的平均值。

网络中影响的传播就像信号在耦合线性系统中传播一样。如果网络是一种 Atay 网络，节点就取间隔［-1，+1］上的值（代表同意或者不同意），或者之间的某一分数。正如影响的传播一样，每个节点值会更改。如果所有节点都到达同一值，经过一段时间后，我们就说网络达成了一致性。如果节点永远达不成一致性，我们就认为网络是发散的。

我们这里要解决的问题是，“在什么条件下影响网络会达成一致性?”这种问题与更加普遍的问题相关，如在何种条件下一个任意网络会同步。我们证明当影响网络状态矩阵的最大非平凡特征值受限于 1 时就能够达成一致，那么在网络中就没有冲突。奇怪的是，网络中最有影响的节点是那些受其他节点影响较少的节点。

网络科学在市场营销领域及理解公司间的竞争中已经取得了更新的成果。我们阐明了在简单随机网络中的偏好连接服从 Bass 和 Fisher-Pry 方程，但是技术扩散在多产品、多竞争网络中更加复杂。偏好连接、价值定位、早期或晚期市场的简单模型以及这些竞争模型的组合导致了通常的垄断以及在某种条件下的寡头垄断。详细讲来，我们证明在大多数情况下紧俏产品的提供者可能与垄断并存。这是一个仍旧需要更多探索才能成熟的研究领域。

有关网络科学简短历史的更详细和更深入的历史事件分析，建议读者进一步学习一下网络科学的几个很精彩的调查。Virtanen 调查了 2003 以前的全部领域（Virtanen，2003）。Voon Joe Yung 的论文给出了非数学的介绍，并且包括一份米尔格兰姆实验用的文件夹（Yung，2001）。Mark Newman 和 Duncan Watts 的大量论文为小世界基础提供了简要的指导（Watts，1999a；Newman，2000b）。

1.3 总则

“现代阶段”通过综合几个有关补充和相互交叉的领域所定义——当然还会有更多即将出现的领域。图论、社会网络分析、流行病建模、市场营销竞争建模以及物理和生物系统的同步都是

网络科学的不同方面。实际上，目前总结或确定网络科学的规范或许还太早，可能会有些目光短浅，以下是作者的看法。

现代阶段的网络科学属性

1. 结构。网络具有结构，它们不是节点和链路的随机集合。例如，电网、在线社会网络以及秀丽小杆线虫（C. elegans）的神经系统结构都不是随机的，而是有着独特的形式或拓扑。这就预示着功能服从形式——许多实际现象的行为方式是由它们的网络结构所决定的。

2. 涌现。作为动态网络取得稳定性的一个结果，如果它的变化是其他的 10 倍，那么该网络属性就涌现。换句话来讲，涌现是网络同步问题——稳定网络从一个状态迁移到另外一个状态直到它们到达某一固定点为止，然后就停留在那里。固定点是对网络的某一属性的相应数量级更改的重新配置。例如，在互联网上，一组青少年要比纯随机行为的一组人高 10 倍的期望值形成小集团，因为小集团是通过偏好连接形成的；排在前面 1% 的美国人要比美国人平均收入高 10 倍；少数影星的受欢迎度是普通影星的 10 倍；世界上少数最大的城市规模要比平均城市大 10 倍。上述每一个例子中，“hub 属性”或向心性是在网络取得一个新的固定点时某些不稳定因素通过网络的结果。这是开始时一无所有，结束时却具有上百万的订阅者的在线社会网络背后的推动力。但是，并不清楚是什么成分加入到在线社会网络后造成了爆炸性的增长。与此类似，不清楚什么动力造成了网络属性的数量级的变化。

3. 动态特性。网络科学与网络结构及动态行为有关。动态行为经常是涌现的结果，或者是导致系统的固定点最终状态的一系列进化步骤的结果。互联网、许多生物系统、某些物理系统和大多数社会系统是增长和变化的网络。人们必须理解它们的动态属性以便全面理解这些系统。仅分析它们的静态结构（如度序列）对于理解网络是不够的。例如，网络同步（如蟋蟀鸣叫的情况下），就是每个蟋蟀的动态行为的结果，也是蟋蟀社会网络的结构（三角子图）。

4. 自治。通过“自愿地”到一起（链路）的自治和自发行为的独立节点形成网络，而不是经过集中控制或集中规划形成网络。结构和功能来自于混沌，更多来自意外新发现而非决定论。例如，大的企业集团的形成是通过小公司的合并而来；大城市的涌现来自小的社区；全球电信系统的形成来自于许多小的、本地的、独立的运营商。网络最初的配置是预谋好的，但是随着时间的推移，网络既可以以某种形式的熵开始“衰变”，也可以经过吸收能量适应或者更改。例如，高速公路系统将会要么衰变并陷入不可修理状态，要么通过努力去修补、延伸、扩容等来改善和提高功能。

5. 自底向上演化。网络是从底部或局部层次上升到达顶部或全局层次的。它们不是自顶向下的设计和实现。这也可以被认为是一种分布式控制，这里网络演变是局部规则应用于局部而没有任何集中控制的结果。即使网络的初始结构是预谋设计好的，网络演变也是它们的动态行为的结果。“未经规划的系统”比如互联网是从局部网络形成的，电网是从局部设施形成的，公路系统是从局部道路和兽道形成的。

6. 拓扑。网络的体系结构或拓扑是一种随着时间涌现的属性，作为一种难以捉摸的力量或节点的自治行为，经常是分布式的。如果其拓扑或其他属性随着时间函数而改变，网络就是动态的。如此一来，拓扑（结构）就是达尔文进化力量形成网络的结果。例如，无标度网络（具有主导 hub 的网络）是在“偏好连接”（经济学）无目的结果（调节律）的力量下出现的，比如电网出现的弱点是政府放松管制的结果，或从复杂自适应系统实现分散的体系结构的“隐秩序”，如互联网的出现、都市文明的形成或像微软公司那样的垄断产生。

7. 有用性。节点的有用性与其度（将节点连接到网络上的链路数量）的影响（链路值）以及介数或紧度成正比，网络的有用性与其节点、链路数和强度有关。例如，梅特卡夫定律定义网络的有用性与其包含的节点数的平方成比例（例如，n 个节点网络的最大链接数可以为 $n(n-$

1)/2，这接近于 n^2）。一个人对一群人施加的影响与人在群内的位置、数量及同事的有用性成正比，例如人的连通性。公司在业界和市场中的有用性与客户（链路）的数量或其在业界的中介位置成比例。有用性难以捉摸，但是在大多数网络中它是重要的组织原则，它经常被表述为其他的形态，如影响、信号强度或者感染率。

8. 稳定性。如果节点/链路改变速率或其拓扑既可以随着时间的推移减少，也可以在有限的限制内受限于阻尼振荡，那么动态网络就是稳定的。例如，动物心脏有规律和节奏地跳动，是受到用以调节节拍器的稳定的神经网络所控制；电网失去一个电厂，会通过快速地从一个源切换到另一个源而很快稳定下来，但不会中断供电；缺少合作者会造成短期的责任重新分配，而不会导致公司组织的停止运营。

本书首先从拓扑的视角，其次采用动态或涌现的观点切入到网络科学的主题。拓扑会满足功能，还是刚好与此相反呢？一个动态网络会因为它的拓扑以某种方式工作或起作用吗？或者它从功能可以导出拓扑吗？接下来的几章提供了很多证据支持“形式服从功能”的观点。稀疏小世界网络似乎建模成人类社会网络，因为人了解很多其他人的能力是有限的。无标度网络是为经济结构（如互联网和工业部门内部的垄断）建模的，因为偏好连接实质上是经济报酬增加律。网络倾向于以这种方式结构化，它更像同步而非混沌，就是因为不稳定的系统不可能在自然界中存在。

那么从底层基础去接近网络科学领域似乎更符合逻辑。首先我们回顾一下图论基础。这需要先给出定义和术语（参见第2、3章）。接下来，生成随机、小世界、无标度和任意拓扑的网络将在第4～7章中论述研究。第8章将流行病学扩展到了网络上，第9章展开讨论网络稳定性的统一理论，第10章将该理论应用到社会网络分析和群组达成一致性上来，第11～13章详细地探索了应用。当然还有更多章节的书写有待于读者来完成。

图

图论是网络科学的历史基础，因此，在网络科学学习之初，我们将重点研究图论。本章内容是随后章节中网络属性分析的必要背景知识，包括集合论和图属性的代数定义，如路径长度属性、熵以及本书其余部分使用的图的分类。具备这方面知识的读者可跳过本章。

图由节点和链路及定义节点如何互连的映射函数所组成。在静态图中，节点、链路和映射函数的属性不随时间而改变。若保持节点数和链路恒定，映射函数就不会变化。在动态图中，节点和链路的数目、映射函数的形式以及图的其他属性随时间的变化而变化。例如，分配给节点和链路的值，以及节点和链路的数目都可能会改变。本章主要集中研究静态图的属性。动态图会在随后的几章中介绍。

我们学习图论的基本定义，介绍其基本属性，并开始设计一组有用的、在随后章节中使用的Java编程方法，以便于仿真网络的构建和行为。

本章定义如下内容：

1. 图。一个图是一个3元组（tuple）：$G=(N,L,f)$，其中N为节点集合，L为链路集合，f为将链路映射到节点对的映射函数（表）。

直接由链路连接的节点称为邻接节点。静态图永远保持它的最初结构。动态图则随着时间的推移更改其节点、链路和/或映射函数。

2. 图的属性。图有很多属性，例如平均路径长度、密度、熵、聚类系数、介数和紧度、谱半径、谱隙、度序列分布。节点度等于将节点连接到图上的链路数量。路径长度是两个节点间的最短路径，平均路径长度是所有最短路径的平均值。密度是实际链路数与最大可能链路数之比。熵是映射函数f中的随机性测量。聚类系数是对“有多少节点与它们的邻接节点形成三角子图”的一种测量。介数和紧度是对一个节点的中介或中间者的有用性的测量。度序列分布和熵是对按照其结构分类图的测量。

3. 矩阵表示。图的映射函数f由包含将节点i和节点j连接起来的链路数的连接矩阵表示。这个数值存储在矩阵的第i行和第j列。在图的邻接矩阵中忽略重复链路——如果一条或多条链路将节点i和节点j连接起来，就在行i和列j处包含一个1。图的拉普拉斯矩阵是通过将节点的度插入到邻接矩阵的对角线上形成的。邻接矩阵和拉普拉斯矩阵的谱分解对于深入了解图的本质非常有用。例如，谱半径是图的邻接矩阵的最大非平凡特征值，而谱隙是图的拉普拉斯矩阵的最大非平凡特征值。图的谱属性用于行为建模，如疾病传染、流言的传播和群组内一致性的达成及混沌在整个图中的传播。

4. 图的分类。我们按照其结构分类，随机图具有随机结构，k - 规则图具有纯的确定性的结构。在这两种极端之间存在两种重要的图：小世界类（大部分结构化的，部分随机的）图，无标度类（大部分随机的，部分结构化的）图。每一类都有自己的属性：随机图服从泊松分布，无标度图服从幂律分布，小世界图具有高平均聚类系数，k - 规则图具有0熵值。

5. 哥尼斯堡七桥。哥尼斯堡七桥问题最早是由欧拉提出，它利用图的基本属性进行分析。结果证明，图论历史中的第一张图若不重复经过一条或多条链路就不可能实现遍历，因为在哥

尼斯堡七桥图中所有节点的度都是奇数值。因此，旅行者必须至少经过一座桥两次才能遍历整张图。

6. *建模和仿真*。在本章最后一节，我们为构建图处理软件引入了 Java 编程语言类。使用简单的数据结构为节点和链路建模，并给出几种简单方法生成链路到节点的随机映射。这些类和方法可以用来理解本书中所给的软件，或者用以构建自己的软件。

本章的例子可以使用开源程序 Network 重新生成并加以分析。所有本书附带的 .jar 程序都是独立的程序，可以在任何支持 Java 运行环境的计算机上运行。用户的操作文件需要使用 PDF 阅读器才能打开。

2.1 图的集合论定义

图 $G=[N,L,f]$ 是一个 3 元组，由一组节点 N、一组链路 L 和一个映射函数 $f: L \rightarrow N \times N$（它将链路映射为节点对）组成。网络 $G(t)=[N(t),L(t),f(t):J]$ 是一个由一组随时间变化的节点 $N(t)$、一组随着时间变化的链路 $L(t)$ 以及随着时间变化的函数 $f(t)$ 构成的三元组。我们称之为微观规则的动态行为是由运算法则 J 来表示，这在本书中就是一组 Java 方法。

在网络中 $N(t)$、$L(t)$、$f(t):J$ 的数量和属性允许随时间变化。从这个角度来讲，网络是一种动态图。例如，网络的形状由行为和映射函数 $f(t):J$ 定义而成，它随着链路的删除、添加或重联（从一个节点切换到另外一个节点）而变化。或者说，网络会随着时间推移因节点和链路的添加而增长。这些随时间变化的特点使得网络有别于图。我们将在下一章中继续研究动态图——网络。

让我们忽略 $G=[N,L,f]$ 中元素随着时间变化的可能性，并且假定 N、L、f 在整个 G 生命周期中保持恒定不变。这样一来，N、L 元素的数量和属性保持固定，而且一旦图建立后映射函数 f 就不再变化。元素 N、L 是什么，以及图论是如何为网络科学提供基础的？这些问题的答案要根据具体提问的领域而定。图论是通用的，可以解释社会学、物理学、商业、生物学和计算机科学中的广泛现象。

2.1.1 节点、链路和映射函数

在社会科学中，N 是个体集合，L 是定义个体间的某种关系的集合。例如，N 可以是一组在一个办公室中一起工作的人员，L 可以是组织结构图中的线条。当然 N 也可以是一组能够传播疾病的人员，L 可以是导致疾病传染的交互。

在物理学和数学中，N 标示顶点、点、节点或代理，而 L 标示边、链路或者 N 个元素之间的连接。例如，N 可以是晶体结构中的原子，L 可以是将结构保持在一起的键。在计算机工程中，节点可以是晶体管，而链路可以是将它们连接起来的导线。在这种情况下，图就是一条电路。另外一方面，N 可以是地图上的城市，L 是将城市连接起来的道路或高速公路。图是一种抽象，这就是我们为什么可以将它们应用到各种不同实际情况中的原因。

在计算机科学中，节点和链路构成了数据结构，用来定义存储和处理的网络。节点和链路可以表示任何事情，因为数据结构是存在于软件之内的抽象实体。因此图和网络是社会、物理和数学结构的模型。在一个例子中，由计算机程序存储和处理的图可能建模成一个社会网络、物理结构、路线图或传染病的扩散途径。在另一个例子中，软件可能模拟穿过道路网络或城市的交通流量、物质中原子键的行为，或者是传染病的传播。图因其通用性而成为一种强大的数据结构。

图是模型，因此网络科学的研究就是一种建模活动。但是要记住模型与实际之间的区别很重要。网络科学就是研究实际建模的，但却不是实际本身。我们需要解答的一个重要问题是："网络建模表示实际系统的程度如何？"在大多数情况下，我们讨论模型如何能与实际很好地拟合。不过，我们必须了解网络模型能不能代表一个实际系统的实际情况。

在本书中，N 的元素称为节点，L 的元素称为链路，映射函数 f 称为 G 的拓扑。N 集的势或大小，标记成小写的 n，它表示 N 中的节点数，而 m 为 L 中的链路数量。

在数学中，$G = [N,L,f]$ 是一个由以下三个集合构成的图：

$N = [v_1,v_2,\cdots,v_n]$ 为节点，$n = |N|$ 为在 N 中的节点数量；

$L = [e_1,e_2,\cdots,e_m]$ 为链路，$m = |L|$ 为在 L 中的链路数量；

$f:L \rightarrow N \times N$ 将链路映射到节点对。

我们使用 v 和 e 分别指示 N 和 L 的元素，并用下标来枚举：$v_1,v_2,\cdots,v_n$，$e_1,e_2,\cdots,e_m$。进一步，符号 ~ 和 → 用来标示将一条链路映射到节点对上，例如，$u \sim v$（无向链路），$u \rightarrow v$（有向链路）。中括号 [ ] 或大括号 { } 标示一个集合，而“:”标示一种映射。

图 2-1 演示了由 $n = 3$ 个节点、$m = 2$ 条链路组成的图 G 和 G'，映射函数 f 和 f' 分别对应于 G 和 G'。对于 G，有

$$f = [e_1:v_1 \sim v_2, e_2:v_2 \sim v_3]$$

G 包含两条链路——e_1 和 e_2。链路 e_1 将节点 $v_1 \sim v_2$ 连接起来，链路 e_2 将节点 $v_2 \sim v_3$ 连接起来。注意这种映射等价于逆向链路映射（因为图是无向的）：

$$f = [e_1:v_2 \sim v_1, e_2:v_3 \sim v_2]$$

这些映射是等价的，因为映射与节点顺序无关——置换一个节点对不会改变 G 的拓扑。当节点对顺序与链路节点对无关时，G 就是一个无向图。在无向图中，$v_1 \sim v_2$ 等价于 $v_2 \sim v_1$。图 2-1a 显示了通过以下映射定义的无向图 G 的拓扑：

$$e_1:v_1 \sim v_2 \text{ 或 } e_1:v_2 \sim v_1$$

$$e_2:v_2 \sim v_3 \text{ 或 } e_2:v_3 \sim v_2$$

作为对比，G' 是一个有向图，因为 $v_1 \sim v_2$ 不等价于 $v_2 \sim v_1$。交换一个节点对的节点的确使图的拓扑有所不同。根据定义，有向图中的链路是有向的，由节点对 (v_1,v_2) 定义的链路与由节点对 (v_2,v_1) 定义的链路是不同的。事实上，两种链路可能存在于一个有向图。因此，图 2-1b 中的 G' 有一个稍微不同的映射函数 f：

$$e_1:v_2 \rightarrow v_1$$

$$e_2:v_2 \rightarrow v_3$$

有向图包含有向链路，即一端是尾部，另一端是头部。例如，在映射 $e_1:v_2 \sim v_1$ 中节点 v_2 是尾部，v_1 是头部。一般来讲，映射函数 f 以尾部 - 头部节点对定义一个有向链路：

$$e:\text{尾部} \rightarrow \text{头部}$$

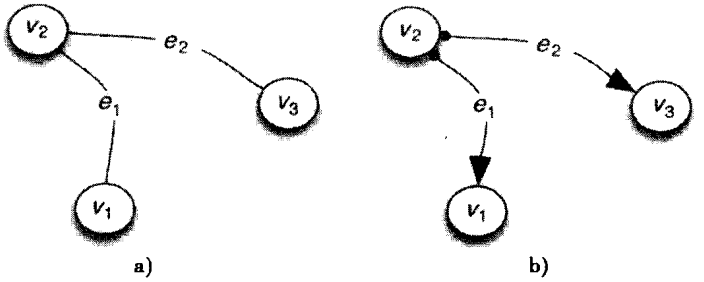


图 2-1 a）无向图 $G = [N,L,f]$ 包含三个节点 v_1、v_2 和 v_3；b）有向图 G' 包含两条链路 e_1、e_2

2.1.2 节点度和 hub

将一个节点 v_i 连接到图的有向或无向链路数量称为节点的度，标记成 degree(v_i) 或 $d(v_i)$ 简写为 d_i。在图 2-1a 中，节点 v_2 的度为 2，其他两个节点的度为 1：

$$d(v_1) = d_1 = 1\ ,\ d(v_2) = d_2 = 2\ ,\ d(v_3) = d_3 = 1$$

当图有向时，如在图2-1b中的G'情况下，节点的出度等于向外方向的链路数，入度等于进入方向的链路数。出度为尾部的数量，入度为连接到节点的头部数量。因此G'的出度和入度映射为：

$$\text{in\_d}(v_1) = \text{in\_d}_1 = 1\ ,\ \text{out\_d}(v_1) = \text{out\_d}_1 = 0$$
$$\text{in\_d}(v_2) = \text{in\_d}_2 = 0\ ,\ \text{out\_d}(v_2) = \text{out\_d}_2 = 2$$
$$\text{in\_d}(v_3) = \text{in\_d}_3 = 1\ ,\ \text{out\_d}(v_3) = \text{out\_d}_3 = 0$$

一张图的hub是具有最大度的节点。节点v_2为图2-1中图的hub，v_2为图2-2a中的hub。简单地计算添加到每一个节点的链路数就可以很容易地确定一个hub节点。数学上，

$$\text{Hub} = \max\text{imum}\{d(v_i)\}$$

一张图可以包含多个hub，因为可能会有多个节点获得同样（最大的）数量的链路。

2.1.3 路径和回路

如果节点u连接到节点v，v连接到节点w，通过传递闭包，节点u连接到节点w。更正式的标记为：

$$\text{Connected}(u,v) = \textbf{true}, \text{if } u \sim v; \textbf{false}\ ,\ \text{otherwise}$$
$$\text{Connected}(u,w) = \text{Connected}(u,v)\ \textbf{and}\ \text{Connected}(v,w)$$

路径是G中一系列连接起来的节点。例如，在图2-1a中节点v_1和节点v_3之间存在一条路径：遍历链路e_1和e_2并在路径上访问节点v_2的路径。一般来讲，如果

$$\text{Connected}(u,v) = \textbf{true}$$
$$\text{Connected}(u,v) = \text{Connected}(u,v_1)\ \textbf{and}\ \text{Connected}(v_1,v_2)\cdots\ \textbf{and}\ \text{Connected}(v_i,w)$$

那么就在起始节点u和终止节点w之间存在一条路径。

路径长度等于路径起始节点和终止节点之间的链路数。路径长度t是以跳数测量的，即沿着路径的链路数量。沿着路径，两个节点间的距离等于将它们分开的跳数。

在图2-1a中从起点v_1到终点v_3的路径长度为$t = 2$跳。但是，在图2-1b中从v_1到v_3没有路径，因为e_1和e_2是有向链路。因此，Connected(v_1,v_2)和Connected(v_3,v_2)两者都是false。但是，有两条路径，每条长度$t = 1$跳，因为Connected(v_2,v_1)和Connected(v_2,v_3)两者都是true。

图中很可能包含连接节点的多条路径。例如，Connected(u,v)可能为true，由于从u到w然后再到v的一条路径，也可能由于从u到s、r、q再到v的第二条路径。第一条路径长度为2跳，第二条路径长度为4跳。一般采用最短的路径作为连接节点u和v的路径。这又称为两个节点间的有向路径。在这个例子中，有向路径的长度为2跳，因此我们就讲u和v之间的距离为2跳。

G的平均路径长度等于所有有向路径的平均值。该度量又称为G的特征路径长度。我们通过将所有有向路径加起来除以路径数来计算G的平均路径。图2-1a的平均路径长度为$(1 + 1 + 2)/3 = 4/3 = 1.33$跳。

开始并结束于同一节点的路径被称为回路。如果回路存在，即从节点u又回到自身，那么Connected(u,u)就为true。循环的回路长度为1，意味着节点u连接到自身。图2-2a既演示了回路（从节点v_2到v_3再回到自身）又演示了循环（从节点v_3到它自身）。当有多条链路连接节点对时，在G中就存在重复链路，也就是在图中有两条或更多相同的映射$v \sim w$。一般来讲，我们讨论的是在节点对中不包含循环和重复链路的图。实际上，我们将尽力避免网络中出现循环和重复，因为在大多数应用中它们没有太大用途。

2.1.4 连通性和组件

沿着从每一个$v_j \neq v_i$的节点（其中$j = 1,2,\cdots,i-1,i+1,\cdots,n$）开始的路径，如果每个节

点 v_i 是可达的，那么无向图 G 是强连通的。图 2-1 中的两个图是强连通的。如果我们忽略图 2-2a 中链路的有向性，那么它也是强连通的。但是如果我们不忽略链路的有向性，那么从节点 v_4 就没有到其他节点的路径。因此，连通性概念在无向图和有向图中的应用是不同的。

如果从 G_1 中的节点到任何 G_2 中的节点之间无（无向）路径存在，那么图 G 具有组件 G_1 和 G_2。换句话讲，组件就是一个独立的子图。图 2-2b 演示了具有两个组件 G_1 和 G_2 的图。注意因为在两个组件之间不存在链路，从 G_1 的一个节点不可能到达 G_2 的一个节点。我们一般仅对强连通的图感兴趣，当从任何一个节点开始都可能遍历整张图时，就简单地称之为连通的。

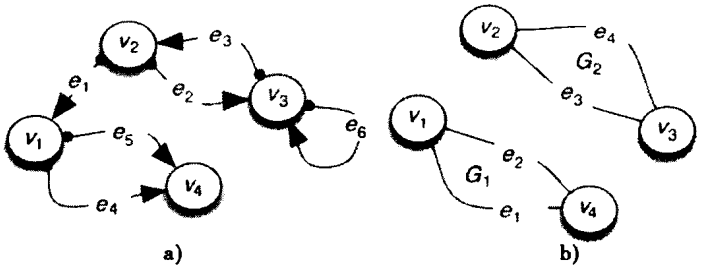


图 2-2 a）节点对 $v_1 \rightarrow v_4$ 和 $v_2 \rightarrow v_3$、$v_3 \rightarrow v_2$ 间有重复链路的和循环的有向图；
b）无向的、不连通的带有两个组件 G_1 和 G_2 的图

2.1.5 直径、半径和中心性

在图 G 中任何两个节点之间最长的路径称为图 G 的*直径*，标记为 diameter(G)。从一个节点 u 到连通图的所有其他节点的最长路径定义成节点 u 的*半径*，标记为 radius(u)。那么具有最小半径的节点为图的*中心*，所有节点中的最大半径就是图的直径。图的直径也等于 G 中最长有向路径的长度。

例如，如果我们忽略图 2-2a 中链路的方向，每个节点的半径为

节点	半径（节点）
1	2 跳
2	2 跳
3	3 跳
4	3 跳

因此，由于 G 中最大的半径为 3 跳，diameter(G) = 3。有两个中心——v_1 和 v_2，因为它们具有 G 中所有节点的最小半径（2 跳）。我们根据离最小半径节点有多少跳来排行每个节点的中心性。在本例子中，节点 1 和 2 是最中心的。

当计算节点 u 的半径时，使用节点对 $u \sim v$ 之间的有向路径。那么就记录最长的（有向）路径作为 u 的半径。更正式地，我们就得到 $\text{radius}(u) = \text{maximum}_i\{\text{minimum}_j\{\text{path}_j(u_i,v_i)\}\}$。

让 $\text{path}_j(u,v_i)$ 等于节点 u 和节点 v_i 之间的第 j 条路径，$\text{minimum}_j\{\text{path}_j(u_i,v_i)\}$ 为 u 和 v_i 之间对于所有 $j = 1,2,\cdots,k$ 条路径的最小长度路径。接下来，将 maximum_i 定义成节点对 (u,v_i) 中最长的有向路径。通过首先计算从 u 到 v_i 节点的最短路径，对于所有 $i = 1,2\cdots,n$，可以找到节点的半径。那么我们就将这 n 条最短路径中最长的那条指定为节点的半径。

图的*直径*是所有这些半径中最长的，图的*中心*是具有最小半径的节点。没有哪个节点比中心节点更接近所有其他节点。没有哪个节点离所有其他节点的距离大于图的直径。具有最大的半径的节点称为*边沿节点*。

图 2-2a 的平均路径长度是多少？为了得到答案，计算所有节点对间的有向路径，然后进行

平均。我们将这种计算组织成一张表（矩阵），使用4行和4列对应于4个节点，如下所示：

		Head			
		v_1	v_2	v_3	v_4
	v_1	–	1	2	1
Tail	v_2	1	–	1	2
	v_3	2	1	1	3
	v_4	1	2	3	–

该表的元素为G中从每一个节点到每一个其他节点的路径长度。行对应于起始节点，列对应于终止节点。例如，从起始节点v_3到终止节点v_1有向路径长度为2跳，从v_4到v_3有向路径长度为3跳。在该表中共有13条路径。所有元素的总和等于21，因此平均路径长度为21/13 = 1.62跳。

在该例子中平均路径长度小于中心节点的半径或图的直径。这样有意义吗？直径为最大值，而平均路径长度为平均值。这样一来，平均路径长度不比整张图的直径大。但是，如果有不相称数量的路径比中心节点半径长，那么就有可能使平均路径长度超过半径。

图2-2b中的不连通图的直径等于多少？为了得到一个路径长度，一个节点必须沿着路径连接到另一个节点上。如此一来，图2-2b的直径为1，因为一跳为所有实际路径的长度。我们忽略在独立的组件中的节点间没有路径的实际情况。

2.1.6 介数和紧度

节点v的介数为从所有节点（除了v）到所有其他节点的必须通过节点v的路径数量。节点v的紧度是从所有节点到所有其他节点的必须通过节点v的有向路径数量。介数和紧度用于度量中间节点的有用性。如果在图2-3中不经过节点v_2就无法从节点v_1向节点v_3发送信息，那么节点v_2就具有超过v_1和v_3的有用性。因为它是一个中介。

介数考虑所有的路径，而紧度仅考虑有向路径。事实上，如果我们在无向图中沿着路径计算两个方向，节点对之间的路径数目可以非常大。正是因为这种原因，我们更愿意使用紧度属性，因为它仅计算有向路径数。这可能仍旧是一个很大的数，但是要比计数所有路径要小得多。例如，比较图2-3中介数和紧度的路径计数：

节点	介数	紧度
1	6	0
2	6	4
3	0	0
4	2	0

注意邻接节点的介数和紧度为0，因为源和目的地节点不是作为过渡节点计数的。介数计数两个路径$v_3 \sim v_2 \sim v_1 \sim v_4$和$v_3 \sim v_2 \sim v_4$，而紧度仅计算短的路径$v_3 \sim v_2 \sim v_4$。

因为其简洁性，我们将使用紧度作为本书其余部分介数的测量方式。进一步来讲，我们标准化路径计数以便于可以在不同的图中比较介数。对区间［0，100］的标准化是通过将路径计数除以最大路径计数然后再乘以100进行的。例如，图2-3的节点v_2在大多数节点之间，因此我们将它指派为中间节点。接下来，我们通过将每个节点记录的路径数除以最经常访问中间节点记录的最大路径数，标准化所有路径计数。在图2-3中，中间节点是v_2，共有四个有向路径通过它，因此我们将所有路径除以4，并将结果表示成百分比：

$$\text{Closeness}(v) = \frac{\text{有向路径数}(v)(100\%)}{\text{有向路径数(中间节点)}}$$

标准化通过图 2-3 的节点的有向路径计算结果：

节点	标准化紧度（%）
1	0
2	100
3	0
4	0

不可能将其他节点沿着有向路径连接起来而不通过节点 v_2。这将节点 v_2 放置到令人羡慕的网守位置——一个假定相对于其他节点很重要的位置。

这里所定义的紧度，并非是测量中间节点有用性超过其他节点的最佳方法。考虑具有偶数个节点的对称图的情形。在两个任意选择的源和目的地节点之间很可能存在几条最短路径，在确定紧度的情况下，计算哪条路径是最短的？在程序 Network. jar 中，通过仅计算第一条找到的有向路径来计算紧度，而非所有的有向路径。当在计算偶数和奇数对称图的紧度时，细心的读者将观察到这种异常。

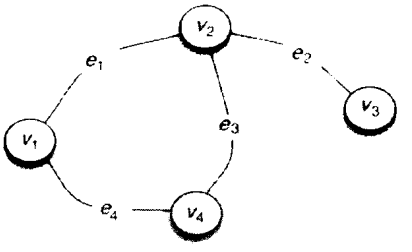


图 2-3 没有循环、无重复链路以及回路长度为 3 的无向连通图 G

2.2 图的矩阵代数定义

集合论对于定理证明和严格地定义图的属性特别有效，但是对矩阵表示的某种形式的分析会更加有效。例如，矩阵表示使得 G 的计算存储和处理都很便利，也可以构成研究 G 结构的数学基础。在很多情况下，矩阵表示是计算图属性的最简洁和最有效的方法。

下面我们学习四种基本形式的矩阵表示：连接矩阵、邻接矩阵、路径矩阵和拉普拉斯矩阵。*连接矩阵*将图的映射函数 f 表示成包含节点对之间链路数的矩形矩阵。*邻接矩阵*是类似的，如果在节点对之间包含一条或多条链路时，邻接矩阵包含一个 1；否则它就包含一个 0。*路径矩阵*的元素等于分隔节点的路径长度，如果不存在路径时则为零。*拉普拉斯矩阵*将节点邻接和节点度信息结合到一个简洁数学公式中，它沿着邻接矩阵的对角线元素存储每个节点的度。

2.2.1 连接矩阵

对于属于 $N \times N$ 的所有节点的所有连接 $v_i \sim v_j$ 的集合 f 定义 G 的拓扑图。这种图是可见的，如图 2-1 和图 2-2 中所示，表格列出的链路和节点对同集合论中一样，或者更加代数地，如同连接矩阵 $\boldsymbol{C}$ 那样，标记为 $\boldsymbol{C}(G)$，这里 $\boldsymbol{C}$ 就是映射函数 f 表示成的矩形矩阵，其中行对应于尾部节点，列对应于 G 中每条链路的头节点。$\boldsymbol{C}$ 的 n^2 个元素设置成连接 G 中的头尾节点对的链路数。

$$\text{如果 } v_i \sim v_j, c_{i,j} = k; \text{ 否则, } c_{i,j} = 0 \quad (i,j) = ([1,n],[1,n])$$

在该定义中，k 为连接 $v_i \sim v_j$ 的链路数。典型地，$k = 1$，但是如果两条链路连接 $v_i \sim v_j$，那么 $c_{i,j} = 2$。例如，在图 2-2b 中，$c_{2,3} = c_{3,2} = c_{1,4} = c_{4,1} = 2$，因为在每个组件中重复链路连接同一节点对，组件是无向的图。

G 的可视转换是从其连接矩阵构建的，反之亦然。例如，图 2-1a 和图 2-1b 中图的各自连接矩阵分别为 $\boldsymbol{C}(G)$ 和 $\boldsymbol{C}(G')$：

$$\boldsymbol{C}(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$C(G') = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \end{array} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

矩阵 C 的每一行对应于链路的尾部，每一列对应于头部。回顾 $v_1 \sim v_2$ 意味着 v_1 是尾部，v_2 是连接 v_1 到 v_2 链路的头部。如此矩阵的行 1 和列 2 包含一个 1。总的来讲，只要节点 v_i 连接到节点 v_j，就在 C 的第 (i, j) 个元素上放 1。

如果多条链路连接两个节点，那么连接 v_i 到 v_j 的链路数被插入到 C 中的行 i 和列 j。注意无向图的矩阵表示 $C(G)$ 和有向图的矩阵表示 $C(G')$ 之间的不同。$C(G)$ 是对称的，而 $C(G')$ 不是的。总的来说，如果图是无向的，因为所有链路在两端具有头和尾则连接矩阵就是对称的。原因在于一个无向图中 $v_1 \sim v_2 = v_2 \sim v_1$。

2.2.2 邻接矩阵

连接矩阵的修改版本（在图论中称为*邻接矩阵*，社会网络理论中的*社会矩阵*）忽略了图中的重复链路。当我们忽略了节点对之间的重复链路，并研究大多数 G 的基本连通性时，邻接矩阵 A 是用来代替连接矩阵。邻接矩阵假定 $k = 0$ 或者 $k = 1$：

$$\text{如果 } v_i \sim v_j, a_{i,j} = 1; \text{ 否则, } a_{i,j} = 0$$

社会矩阵或邻接矩阵又称为*布尔矩阵*，因为所有项必须要么为 0 要么为 1（布尔值）。如果两个节点连接，邻接矩阵记录一个 1；否则，就记录一个 0。当用计算机操作矩阵 A，这种邻接性质是很有用的。例如，两个邻接矩阵的布尔积就是一个包含 0 和 1 的布尔矩阵的自身。

社会矩阵和邻接矩阵是同一事物的不同名字。在社会网络分析中，图 2-1a 表示一群人中的一种社会关系，例如朋友、情人或管理者 - 雇员关系。假如 Tom（v_1）认识 Mary（v_2），但不认识 Jill（v_3）。图 2-1a 中的图 G 为这种小社会网络建模，并告诉大家 Mary 是三人网络的中心，因为半径（Mary）为 1 跳，而半径（Tom）和半径（Jill）都是 2 跳。

社会网络分析更加关心在图中建模成节点的一对人之间至少存在一种关系。这样一来，社会矩阵实际上是一个邻接矩阵。当分析一个社会矩阵时，沿着重复链路删除了循环。图 2-2a 证明了连接矩阵和邻接矩阵的不同。连接矩阵考虑所有的链路，而邻接矩阵仅考虑邻接——由一个布尔值指示。邻接节点要么被一条要么被多条链路连接（1），要么它们没有连接（0）。图 2-2a 的连接矩阵和邻接矩阵为：

$$C(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 2 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} = \begin{pmatrix} 0 & 0 & 0 & 2 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

注意区别：C 中在行 1 列 4 的位置包含一个 2；而 A 在对应的（1，4）元素包含了一个 1。此外，元素（3，3）包含了一个 1，这是因为在节点 v_3 的循环。连接矩阵列出节点对之间的连接数

量，而邻接矩阵在存在链路数时列出布尔1，否则为0。两种矩阵都是非对称的，因为图是有向的。邻接矩阵表示或许是获取最小拓扑信息的最紧凑表示。正因为此，我们使用邻接矩阵及其扩展表示网络结构。

2.2.3 拉普拉斯矩阵

图 G 的*拉普拉斯矩阵*，命名为 $L(G)$，是连接矩阵和（对角）度矩阵的组合：$L = C - D$，这里 D 为对角矩阵而 C 为连接矩阵。矩阵 D 是一个具有零非对角元素并且对角元素 $d_{i,i}$ 等于节点 v_i 的度的一致性矩阵：

$$d_{i,j} = \begin{cases} \text{degree}(v_i) & \text{如果 } j = i \\ 0 & \text{其他} \end{cases}$$

L 的行元素总和等于0时也为真，如果矩阵对称，那么列元素总和也是如此。这是由于无向链路的对称性：$v \sim u = u \sim v$。当我们使用拉普拉斯研究网络的稳定性时，该事实将在稍后变得更加重要。

图 2-3 的无向图的连接和拉普拉斯矩阵为：

$$C(G) = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ v_1 & 0 & 1 & 0 & 1 \\ v_2 & 1 & 0 & 1 & 1 \\ v_3 & 0 & 1 & 0 & 0 \\ v_4 & 1 & 1 & 0 & 0 \end{array} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

$$L(G) = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ v_1 & -2 & 1 & 0 & 1 \\ v_2 & 1 & -3 & 1 & 1 \\ v_3 & 0 & 1 & -1 & 0 \\ v_4 & 1 & 1 & 0 & -2 \end{array} = \begin{pmatrix} -2 & 1 & 0 & 1 \\ 1 & -3 & 1 & 1 \\ 0 & 1 & -1 & 0 \\ 1 & 1 & 0 & -2 \end{pmatrix}$$

2.2.4 路径矩阵

路径矩阵 $P(G)$ 存储了沿着图中所有节点对之间有向路径的跳数，即 $P(G)$ 枚举了所有节点对中最短路径的长度。如果图是无向的，那么 $P(G)$ 就是对称的。至于邻接矩阵，如果在节点对之间不存在链路，那么在 $P(G)$ 中对应元素为0。$P(G)$ 的对角元素为0，因为我们不考虑循环以及重复链路。

例如，图 2-3 的无向图的路径矩阵是对称的，因为 G 为无向的：

$$P = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ v_1 & 0 & 1 & 2 & 1 \\ v_2 & 1 & 0 & 1 & 1 \\ v_3 & 2 & 1 & 0 & 2 \\ v_4 & 1 & 1 & 2 & 0 \end{array} = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 2 \\ 1 & 1 & 2 & 0 \end{pmatrix}$$

对角元素都为0，非对角包含沿着路径连接每个节点对的跳数。在 v_1 和 v_3 之间有两条路径（$v_1 \sim v_2 \sim v_3$ 和 $v_1 \sim v_4 \sim v_2 \sim v_3$），但是我们仅存储最小的路径长度，因此元素 $p_{1,3}$ 和 $p_{3,1}$ 都等于2跳。类似地，在 v_3 和 v_4 之间存在另外两条路径（$v_3 \sim v_2 \sim v_4$ 和 $v_3 \sim v_2 \sim v_1 \sim v_4$），但是矩阵 P 仅存储最短路径长度。事实上，该图的直径为2跳，因为每一个节点都可以从每个其他节点最多以2跳到达。因此 P 的任意元素可能的最大值为2。

路径矩阵与邻接矩阵之间的关系如下。让 A 为邻接矩阵，P 为路径矩阵。A^2 为包含所有相距为2跳的节点对的相邻矩阵，A^3 为包含3跳邻接，A^k 为包含 k 跳邻接。这样一来，$A,A^2,A^3,\cdots,A^k$ 就依次是路径长度为 $1,2,3,\cdots,k$ 的路径矩阵。注意在集合 $[A,A^2,A^3,\cdots,A^k]$ 中的最短路径为路径矩阵的元素。让 D 为最长路径的大小——G 的直径。那么，$P = \min_{k=1}^{D}\{kA^k\}$ 刚好就是 G 的路径矩阵。这里 min 函数逐个地选择集合中最小的元素。

矩阵 $A,A^2,A^3,\cdots,A^k$ 必须都是布尔值，因此我们通过将非零元素换成布尔值1来简化每个矩阵的元素。每个矩阵在 $A^k = A^{k-1}A$ 中的乘积后，零仍旧为零。kA^k 乘积产生以 k 跳将节点连接起来的路径长度。我们忽略循环，因此每次乘积后对角线元素设置为零[⊖]。非对角元素等于每个矩阵 kA^k 的长度为 k 的跳数。P 存储有向路径长度，这就意味着我们仅将 kA^k 最小的非平凡元素等同对应的 P 元素。

再次考虑图2-3的无向图。其邻接矩阵为

$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}\begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \end{array} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

乘以 A 得到经过 $2,3,\cdots,k$ 跳的节点对邻接。2跳矩阵为 $A \times A$：

$$A^2 = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}\begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 3 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 2 \end{pmatrix} \end{array} = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 3 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 2 \end{pmatrix}$$

零化对角线，将 A^2 简化为布尔矩阵，并乘以 $k = 2$ 得出

$$2A^2 = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}\begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 2 & 2 & 2 \\ 2 & 0 & 0 & 2 \\ 2 & 0 & 0 & 2 \\ 2 & 2 & 2 & 2 \end{pmatrix} \end{array} = \begin{pmatrix} 0 & 2 & 2 & 2 \\ 2 & 0 & 0 & 2 \\ 2 & 0 & 0 & 2 \\ 2 & 2 & 2 & 2 \end{pmatrix}$$

这是路径长度为2跳连接节点的邻接矩阵的两倍。例如，节点 v_1 经过两跳（$v_1 \sim v_4 \sim v_2$）链接到 v_2，节点 v_3 经过两跳路径长度连接到节点 v_1 和 v_4。

在本例子中，直径 $D = 2$，因此我们已经找到了长度为 $1,2,\cdots,D$ 的所有路径。下一步通过选择每个的非平凡最小元素将矩阵结合起来，也就是选择非零最小值：

$$P = \min\ (k = 1,2)\{kA^k\} = \min\{A,2A^2\}$$

$$P = \min \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} 0 & 2 & 2 & 2 \\ 2 & 0 & 0 & 2 \\ 2 & 0 & 0 & 2 \\ 2 & 2 & 2 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 2 \\ 1 & 1 & 2 & 0 \end{pmatrix}$$

邻接矩阵 A 的幂也包含沿着 A^k 的对角长度为 k 的循环。这种属性将在后面章节以多种方式

⊖ 通常，循环长度 k 标示成 A^k 中的非零对角线。

运用到。例如，除了零化 A^k 的对角元素外，我们可以利用它们在任意图 G 中找到长度 $2,3,\cdots,k$ 的循环。这种算法留给读者作为练习。

2.3 哥尼斯堡七桥图

既然我们对图已经有了初步的理解，并知道如何将它们的结构表示成集合论和矩阵代数的形式，下面考虑将该理论应用于实际例子。下面的例子为图论之父欧拉所创建，他使用新的图论解决了一个非常实际的问题。通过该应用，欧拉奠定了现代图论，同时也为网络科学打下了基础。

2.3.1 欧拉路径和欧拉回路

图 2-4a 是欧拉的第一张图——这位著名的瑞士数学家于 1735 年发表的哥尼斯堡七桥问题的模型。回顾上一章内容，哥尼斯堡市民想要知道是否可能穿越整座城市，通过每座桥一次且仅一次，再返回到出发点。返回到出发点的路径称为*回路*。遍历图中所有链路的路径被称为*欧拉路径*，*欧拉回路*是开始和结束于同一节点的欧拉路径。市民们对遍历欧拉回路的要求，使得该问题变得有趣和困难。

城市范围包括 Pregel 河的两岸以及位于 Pregel 河中的两座小岛。七桥系统连接着陆地，如图 2-4a 所示。节点表示四块陆地，七条链路代表连接的桥。节点 v_2 和 v_3 表示 Pregel 河的两岸，节点 v_1 和 v_4 表示两个小岛。两条链路将小岛 v_1 连接到北岸 v_3，另外两条将 v_1 连接到南岸。类似地，另外三座桥（链路）将小岛 v_4 连接到两岸和小岛 v_1，这就是所有的七桥。

我们可以将哥尼斯堡七桥问题改述成图的问题："是否可能沿着图 2-4a 中的整张图行走，从任意一个节点开始再返回到出发点，而不多于一次地经过任何一条链路？"换句话讲，"G 包含一个欧拉回路吗？"欧拉证明了沿着该图行走而不重复任一条链路是不可能的，因此不存在欧拉回路。

欧拉研究的图是无向的并且没有循环，但是的确有重复的链路。因此它的连接矩阵不同于它的邻接矩阵。进一步来讲，最大的半径长度为 2 跳，因此该图的直径为 2。换句话讲，市民可以在不穿越一座桥多于两次的情况下，从任何一块陆地到达任意其他一块陆地。

2.3.2 哥尼斯堡七桥问题的正式定义

欧拉图的集合论定义 $G_{\text{哥尼斯堡}} = [N,L,f]$ 为

$N = [v_1,v_2,v_3,v_4]$

$L = [e_1,e_2,e_3,e_4,e_5,e_6,e_7]$

$f = [e_1:v_1 \sim v_2, e_2:v_1 \sim v_2, e_3:v_1 \sim v_3, e_4:v_1 \sim v_3, e_5:v_1 \sim v_4, e_6:v_2 \sim v_4, e_7:v_3 \sim v_4]$

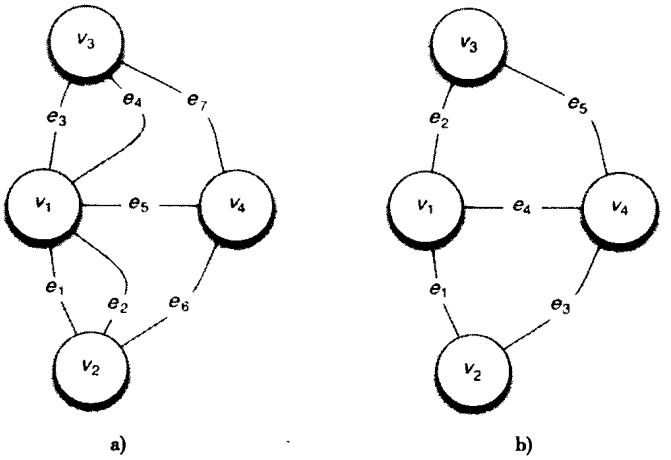


图 2-4 哥尼斯堡七桥图：a）欧拉原先的图包含 $n = 4$ 个节点表示陆地，$m = 7$ 表示桥；b）删除了两条重复链路 e_2 和 e_4 后的简化欧拉图

欧拉图的矩阵代数定义 $\boldsymbol{C}$（$G_{\text{哥尼斯堡}}$）为

$$\boldsymbol{C} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 2 & 2 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array} = \begin{pmatrix} 0 & 2 & 2 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

矩阵 $\boldsymbol{C}$ 的每一行和列对应于一个节点：行 1/列 1 对应于节点 v_1，行 2/列 2 对应于节点 v_2，以此类推。如果 $v_i \sim v_j$，那么就将 v_i 连接到 v_j 的链路数插入到连接矩阵的行 i 和列 j；否则，在行 i 和列 j 交叉处出现一个零。矩阵 $\boldsymbol{C}$ 是对称的，因为链路是无向的：$v_i \sim v_j$ 等于 $v_j \sim v_i$。

类似地，邻接矩阵 $\boldsymbol{A}$（$G_{\text{哥尼斯堡}}$）为

$$\boldsymbol{A} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

邻接矩阵相当于如图 2-4b 所示的简化（删除了重复链路）图。

在图 2-4a 中，从节点 v_1 到节点 v_2 存在很多链路：一条路径经过链路 e_1，另外一条路径经过 e_2，稍长的路径（2 跳）经过链路 e_5 和 e_6。v_1 到 v_2 之间一条更长的路径经过 e_4、e_7、e_6。但是最短的路径定义了 v_1 到 v_2 之间的距离。在图 2-4a 中，v_1 到 v_2 之间的路径为 1 跳：两条长度为 1 的路径连接 v_1 到 v_2 ——一条是经过链路 e_1，另外一条是经过链路 e_2。v_1 到 v_2 之间距离仍为 1 跳。

路径矩阵 $\boldsymbol{P}$（$G_{\text{哥尼斯堡}}$）为

$$\boldsymbol{P} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

邻接矩阵 $\boldsymbol{A}$ 和路径矩阵 $\boldsymbol{P}$ 仅在一个方面不同：v_2 到 v_3 之间的路径为 2 跳，这将导致 $p_{2,3} = p_{3,2} = 2$。既然 v_2 和 v_3 是不相邻的，$a_{2,3} = a_{3,2} = 0$。路径矩阵可以通过选择 $\boldsymbol{A}$ 的幂的最小非平凡元素，从前面显示的邻接矩阵中计算出来：

$$\boldsymbol{P} = \min\{\boldsymbol{A}, \boldsymbol{A}^2\}$$

这留给读者作为练习。

欧拉图具有很多回路。例如 $v_1 \sim e_1 \sim v_2 \sim e_2 \sim v_1$ 是一个开始和结束于节点 v_1 的回路，而 $v_1 \sim e_5 \sim v_4 \sim e_6 \sim v_2 \sim e_2 \sim v_1$ 是另外一个回路。图 2-4a 中包含一个欧拉回路吗？尝试从 v_1 出发并回到节点 v_1 遍历整张图，看看会发生什么？不可能遍历所有链路一次并且仅一次。图 2-4b 的简化欧拉图包含一个欧拉回路吗？

2.3.3 欧拉解

在求解哥尼斯堡七桥问题中，节点度扮演着核心的角色。欧拉证明市民不能沿着欧拉回路穿越整座城市，因为陆地是通过奇数座桥连接起来的。不可能访问一个节点（陆地）而返回一个偶数次，因此沿着欧拉回路，就不可能仅访问奇数度节点一次。注意，在避免遍历链路多于一次时，可能访问节点多于一次，但是节点的度必须是偶数才能如此。

连接每块陆地的桥数（图2-4a中每个节点的度）存储在该图的拉普拉斯矩阵中：

$$L = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} -5 & 2 & 2 & 1 \\ 2 & -3 & 0 & 1 \\ 2 & 0 & -3 & 1 \\ 1 & 1 & 1 & -3 \end{pmatrix} \end{array} = \begin{pmatrix} -5 & 2 & 2 & 1 \\ 2 & -3 & 0 & 1 \\ 2 & 0 & -3 & 1 \\ 1 & 1 & 1 & -3 \end{pmatrix}$$

L 的对角元素等于图2-4a中节点（陆地）的度。这是一个拉普拉斯矩阵，因为行的和等于零。列之和也等于零，因为图是无向的。在哥尼斯堡七桥图中不存在欧拉回路，因为对角线元素为奇数。

欧拉证明了以下任意图的一般结果，特别是哥尼斯堡七桥图：

1. 如果任何节点具有奇数度，连通图没有欧拉回路。根据拉普拉斯算子，所有节点具有奇数的度。因此哥尼斯堡七桥图不会有欧拉回路。直观上讲，欧拉回路有一定意义——遍历每条链路一次并返回到起点，旅行者必须离开每一块陆地的次数与他到达的次数相同，因此就必须有一个偶数条链路连接每个节点。该结果可以应用到图2-4b的简化欧拉图吗？

2. 如果图是连通的，并且每个节点具有偶数度，那么它至少有一个欧拉回路。像前面例子中的理由一样，这种状况保证存在一个欧拉回路。假设图2-4a中的图是重联的，以便替换成 e_5：$v_2 \sim v_3$。现在所有节点具有偶数度，因此，至少一个回路是欧拉回路。找出这条回路留给读者作为练习。

3. 欧拉继续证明了两个以上节点具有奇数度的图没有欧拉路径，并且如果它刚好具有两个奇数度节点，那么至少存在一个欧拉路径开始并结束于奇数度的节点。欧拉路径能让市民遍历所有的桥一次，但是却不能回到起点。这种理论也可以应用到图2-4a吗？也可以应用到图2-4b吗？

4. 图中所有节点的度的总和为一个偶数，等于链路数的两倍：$\sum_{i=1}^{n} \text{degree}(v_i) = 2m$。$G$ 的度的总和可以从 L 的对角线获得。忽略 L 的对角线标记，哥尼斯堡七桥图总的度数为 $5+3+3+3=14$，这刚好是2（7）条链路。类似地，图2-4b的节点度的总和为偶数。这意味着什么？

5. 在每张图中，奇数度节点数必须为偶数，因为 $2m$ 为偶数。再次，我们可以使用存储在拉普拉斯矩阵中的信息对此加以验证。奇数度的节点数为4，这是一个偶数。该结论服从观察到的度的总和为偶数，偶数总是两个偶数或者两个奇数之和。一个偶数和一个奇数之和为奇数。图2-4b有两个奇数度的节点。当应用到图2-4b时，该论断经得起验证吗？

欧拉从一个实际问题的表象出发创建了图论。他是通过严格的分析方法完成的。我们要学习他的例子以及他所使用的严格的方法！当我们在图论中解决大量实际问题时，尽管形式主义可以指导我们，但是我们生活在丰富的、低成本计算功能的时代。这就使得我们能够将实际与理论相结合，并很有希望获得比纯形式演绎更深入的理解。因而，本书的方法是源于形式主义（例如图的拓扑和矩阵代数）和非形式主义（如计算机仿真和涌现）。网络科学不只是图论的应用——它是数学和计算科学的结合，而且包括现实世界系统的应用。

2.4 图的谱属性

我们已经阐述了如何从邻接矩阵和拉普拉斯矩阵的角度定义图。接下来，我们演示如何从这两个矩阵中提取动态属性，使用与理解线性系统行为的方法相类似的技术，如乐器中振动的弦、量子力学中的粒子，或者各种其他机电系统。谱半径是从邻接矩阵中计算出来的，谱隙是从图的拉普拉斯矩阵中计算出来的。将矩阵看成根据它的动态方程 $D[A] = AX$ 随着时间和空间变化的线性系统变换，这里 D 为线性运算符，A 为某种类型的线性变换，X 为系统的状态。例如，D

可能是时间导数运算符，A 可能是作用于机械系统上的阻尼力，例如机器人手臂或汽车减震系统。

典型的（功能良好的）线性系统将要通过某种模式循环。我们可以将系统对刺激的响应分解成一组基本的模式或基本矢量——数学中又称为谱分解，以便确定这种模式的本质。这些基本矢量被称为正交矢量或特征向量。具体说，如果 $\boldsymbol{\lambda}$ 为对角矩阵，那么 $\boldsymbol{A}$ 就可以分解为 $\boldsymbol{A} = \boldsymbol{\lambda I}$，这里 $\boldsymbol{I}$ 是单位矩阵，$\boldsymbol{\lambda}$ 是包含特征值的矩阵，或者 $\det[\boldsymbol{A} - \boldsymbol{\lambda I}] = 0$，这里 det [ ] 是 [ ] 的行列式值。那么

$$\boldsymbol{\lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

冒着过于简化的风险，谱分析是找到线性系统基本振动模式（谐频）并将它们表示成称为特征值的常数的过程。最大的这种特征值一般标记成 $\boldsymbol{\lambda}_1$，具有特别的意义，因为它表示动态系统的主模式。进一步来讲，如果 $\lambda_1 < 0$，振动最终会消失；否则，它就会增加并成为不稳定的系统。

网络科学中使用两种谱测量：谱半径和谱隙。图 G 的谱半径是 G 的邻接矩阵的最大非零特征值，而 G 的谱隙就是它的拉普拉斯矩阵的最大非零特征值。后面我们使用谱半径解释传染病是如何在网络中传播的，并用谱隙来解释网络稳定与否。

2.4.1 谱半径

谱半径 $\rho(G)$ 是 $\det[\boldsymbol{A}(G) - \boldsymbol{\lambda I}] = 0$ 的最大非平凡[⊖]特征值，这里 $\boldsymbol{A}$ 为邻接矩阵，$\boldsymbol{I}$ 是单位矩阵。特征值是 $\boldsymbol{\lambda I}$ 的对角线 $\lambda_1, \lambda_2, \lambda_3, \cdots, \lambda_n$。谱半径特征值又称为属性特征值，因为它们以简明的方式描述图的拓扑。事实上，它们经常是等同于结构图的平均度 λ，或与结构图（见表 2-1）的平均度 λ 密切相关。回顾一下，平均度 $\lambda = (2m/n)$ 是连接到节点的平均链路数。

平均度与谱半径并不完全相同，正如稍后我们在研究随机和无标度图时所看到的那样。目前，将谱半径看做图映射函数 f 的简洁表示法。事实上，谱半径完全由图的拓扑（度序列）来决定，因此是图的拓扑的测量。在表 2-1 中，二叉树的项包含从 $n = 3$ 到 $n = 1023$ 个节点范围的 6 个例子。随着二叉树规模的增加，它的平均度接近 2.0，其谱半径似乎接近于 exp（1）。其证明留给高级读者作为练习。

如何计算一个任意图的谱半径？再次考虑图 2-3 中的图，并构造一个如下的邻接矩阵：

$$\boldsymbol{A} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \end{array} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

接下来，形成矩阵 $\boldsymbol{A}(G) - \boldsymbol{\lambda I}$ 并找到元素 λ，求解从行列式方程 $\boldsymbol{A}(G) - \boldsymbol{\lambda I} = 0$ 获得的多项式，$\det[\boldsymbol{A} - \boldsymbol{\lambda I}] = 0$：

$$\det\begin{bmatrix} -\lambda & 1 & 0 & 1 \\ 1 & -\lambda & 1 & 1 \\ 0 & 1 & -\lambda & 0 \\ 1 & 1 & 0 & -\lambda \end{bmatrix} = 0$$

⊖ 非平凡就是所谓的非零。

沿着第3列使用拉普拉斯展开式展开行列式，就得到如下多项式：

$$\lambda^4 - 4\lambda^2 - 2\lambda + 1 = 0$$

这种多项式的根为 $\{-1.48, -1.0, 0.311, 2.17\}$，最大的非平凡根为2.17，因此 $\rho = 2.17$。注意该图的平均度为2.0，平均路径长度为1.333跳。

表2-1 某些规则图的谱半径（n 为节点数）

图	谱半径（ρ）	平均度（λ）
完全	$(n-1)$	$(n-1)$
超环形	4	4
超立方	$\log_2(n)$	$\log_2(n)$
1-规则（环形）	2	2
星形	$\sqrt{n-1}$	$(n-2)/n$
二叉树	$\sqrt{(2)}; n = 3$	4/3
	$2; n = 7$	1.7142
	$2.2882; n = 15$	1.8666
	$2.4494; n = 31$	1.9354
	$2.548; n = 63$	1.9682
	$2.714; n = 1023$	1.9980

2.4.2 谱隙

拉普拉斯矩阵 $\boldsymbol{L}(G)$ 是一个无向的、无循环的、无重复的图的矩阵，其对角线定义如下：

$$\text{Diagonal}(i,i) = -\sum_{j \neq i} c_{i,j} = -\text{degree}(v_i)$$

假设 $\boldsymbol{A}$ 为邻接矩阵，$\boldsymbol{D}$ 为包含节点的度的对角，$\boldsymbol{L}$ 为拉普拉斯算子。那么 $\boldsymbol{L} = \boldsymbol{A} - \boldsymbol{D}$。$G$ 的谱隙为 $\boldsymbol{L}$ 的最大非平凡特征值。例如，在图2-3中对角矩阵是从标记 $\text{degree}(v_1) = 2$，$\text{degree}(v_2) = 3$，$\text{degree}(v_3) = 1$，$\text{degree}(v_4) = 2$ 获得的。那么从 $\boldsymbol{A}$ 减去 $\boldsymbol{D}$ 就得到 $\boldsymbol{L}$，导致沿着对角线出现负数。以这种方式，$\boldsymbol{L}$ 的每行和列元素的总和等于零，注意如果邻接矩阵是对称的话，拉普拉斯算子就是对称的。

图2-3的图是无向的，并且没有重复或循环链路，因此它的邻接、对角和拉普拉斯矩阵为：

$$\boldsymbol{A} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \end{array} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

$$\boldsymbol{D} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \end{array} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

$$\boldsymbol{L} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} -2 & 1 & 0 & 1 \\ 1 & -3 & 1 & 1 \\ 0 & 1 & -1 & 0 \\ 1 & 1 & 0 & -2 \end{pmatrix} \end{array} = \begin{pmatrix} -2 & 1 & 0 & 1 \\ 1 & -3 & 1 & 1 \\ 0 & 1 & -1 & 0 \\ 1 & 1 & 0 & -2 \end{pmatrix}$$

图的谱隙就是其拉普拉斯矩阵 $\boldsymbol{L}$ 的最大非平凡特征值。我们通过找到 $\det[\boldsymbol{L}-\lambda\boldsymbol{I}]=0$ 的最大非零特征值来计算谱隙 $\sigma(G)$。再次使用图 2-3 的图和一台计算机找到大规模多项式的根，我们就会得到特征值 $\{-4,-3,-1,0\}$。最大非零特征值为 $\sigma=-1.0$。可以获得如下谱隙：

$$\boldsymbol{L}-\lambda\boldsymbol{I}=\begin{pmatrix}-2-\lambda & 1 & 0 & 1\\ 1 & -3-\lambda & 1 & 1\\ 0 & 1 & -1-\lambda & 0\\ 1 & 1 & 0 & -2-\lambda\end{pmatrix}$$

设置 $\det[\boldsymbol{L}-\lambda\boldsymbol{I}]=0$，并且按照第3列使用行列式的拉普拉斯公式展开，产生多项式 $(\lambda^3+8\lambda^2+19\lambda+12)\lambda=0$，根为 $\{-4,-3,-1,0\}$。因此 $\sigma=-1.0$。

某些作者将拉普拉斯算子定义成 $\boldsymbol{L}(G)=\boldsymbol{D}-\boldsymbol{A}$，取反了 L 元素的符号。这里使用的定义与邻接矩阵的定义更加一致，在考虑网络的同步属性时会更加有用。因此我们使用如下定义：保留由邻接矩阵生成的布尔值的符号。

当强调 L 的所有元素必须小于 1 时，就要使用正交化拉普拉斯矩阵。正交化的版本沿着对角线为 1，代替布尔值 (0, 1)，邻接就表示成 $\left(0,\frac{-1}{\sqrt{d_i d_j}}\right)$，这里 $(d_i d_j)$ 为节点度值的乘积。这种版本是将拉普拉斯算子定义成 $\boldsymbol{L}=\boldsymbol{I}-\boldsymbol{D}^{-(1/2)}\boldsymbol{A}\boldsymbol{D}^{-(1/2)}$ 的结果，这里 $\boldsymbol{I}$ 是单位矩阵，$\boldsymbol{D}$ 为对角矩阵（在其对角包含 G 的节点的度）。

拉普拉斯算子有时也定义成 $\boldsymbol{L}=\boldsymbol{D}^{-1}\boldsymbol{A}-\boldsymbol{I}$。这些替代定义没有一个能产生包含行和列之和等于零的矩阵。因为“零总和”特性在研究网络同步属性时非常有用，因此将拉普拉斯算子定义成 $\boldsymbol{L}=\boldsymbol{A}-\boldsymbol{D}$ 或 $\boldsymbol{L}=\boldsymbol{D}-\boldsymbol{A}$ 会更有用。

2.5 图的类型

我们已经研究了几种基本的图结构——节点的度、平均路径长度、连通性和中心性（半径和直径）。但是，我们想要描述图的更多内容，而不仅是这些基本度量。例如，我们想要根据其结构对图进行分类，这最终要由映射函数 f 来确定，并且我们想要研究图的结构对涌现的影响。这就需要对图结构进行大量额外的测量。

在本书接下来的内容中，图由它的节点度的分布、聚类的集中量以及由节点半径分布确定的节点中心性来描述。

2.5.1 杠铃形、线形和环形图

或许最简单的图是杠铃形（参见图 2-5a）。它是线形图或链图的特殊情况，如图 2-5b 中所示。线形图的映射函数定义了一个线性序列节点，每个节点连接到后续节点。序列中的第一个和最后一个节点的度为 1，而中间节点的度为 2。线形图映射函数将 $(n-1)$ 条链路映射到节点对上，如下所示：

$$f_{\text{line}}=[e_i:v_i\sim v_{i+1}];\quad i=1,2,\cdots,n-1$$

环形图具有类似的拓扑，但是链或序列中的终止节点连接到起始节点上。因此，n 条链路将所有节点连接到后续节点上（终止节点“围绕”连接到起始节点上）。所有节点的度等于 2。环形图映射函数为：

$$f_{\text{ring}}=[e_i:v_i\sim v_{i(\bmod n)+1}];\quad i=1,2,\cdots,n$$

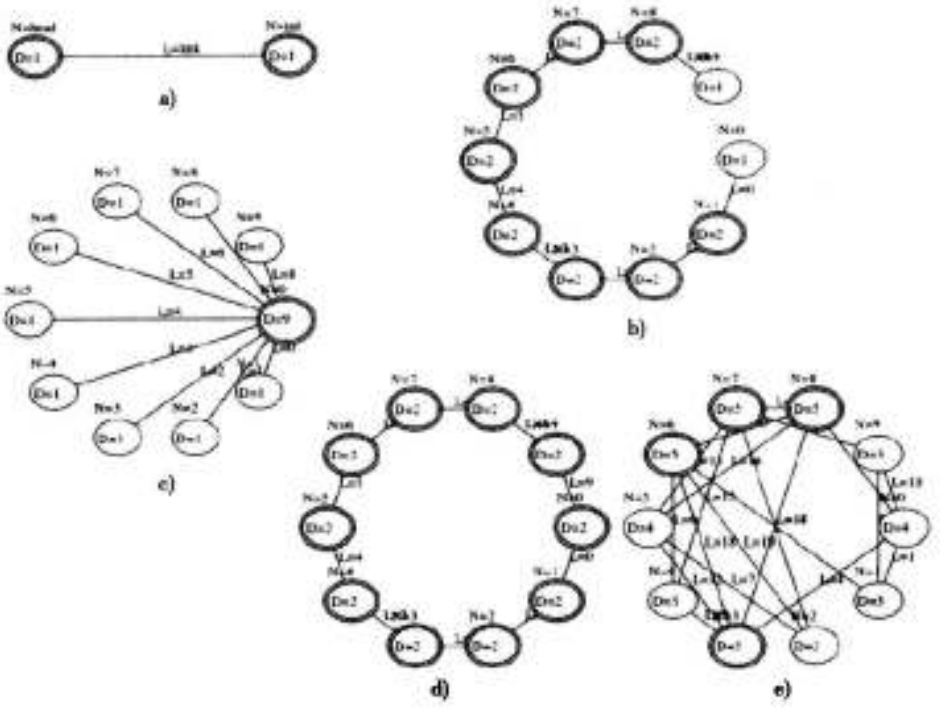


图 2-5 图的类型：a）杠铃形，具有两个节点和一条链路；b）线形，具有 n 个节点和 $(n-1)$ 条链路按顺序将它们连接起来；c）星形，具有 n 个节点，所有节点经过 n 条链路连接到一个节点上；d）环形，与线形相似，但是最后的节点连接到线形中第一个节点上；e）随机，具有 n 个节点和 m 条链路——头部和尾部节点是随机选择的

注意 $i(\mathrm{mod}\ n)+1$ 从最后一个节点围绕到第一个节点。例如，如果 $n=10$，如图 2-5d 中所示，那么 e_{10}：$v_{10}\sim v_1$，因为 $10(\mathrm{mod}\ 10)+1=0+1=1$。

杠铃形、线形和环形图为强连通的，因为从每一个节点到每个其他节点都存在一条路径。但是，仅有环形图才具有回路。这是一个欧拉回路吗？回答是肯定的，因为每个节点具有偶数度。显然，在杠铃形或线形图中没有欧拉回路，但是这些“序列图”中存在欧拉路径吗？

因为环形图映射函数的围绕特征，线形和环形图中的平均路径长度具有很大的不同。环形图中的任意一对节点间存在有两条路径（顺时针方向与逆时针方向），但是在线形图中仅有一条路径。因此，线形图的平均路径长度为 $O(n/3)$，环形图的平均路径长度为 $O(n/4)$。为了理解为什么会存在这种区别，分析 $n=6$ 的线形图和环形图的路径度量：

$$\boldsymbol{P}(\mathrm{line})=\begin{array}{c}\\ v_1\\ v_2\\ v_3\\ v_4\\ v_5\\ v_6\end{array}\begin{array}{c}\begin{array}{cccccc}v_1 & v_2 & v_3 & v_4 & v_5 & v_6\end{array}\\ \begin{pmatrix}0&1&2&3&4&5\\1&0&1&2&3&4\\2&1&0&1&2&3\\3&2&1&0&1&2\\4&3&2&1&0&1\\5&4&3&2&1&0\end{pmatrix}\end{array}=\begin{pmatrix}0&1&2&3&4&5\\1&0&1&2&3&4\\2&1&0&1&2&3\\3&2&1&0&1&2\\4&3&2&1&0&1\\5&4&3&2&1&0\end{pmatrix}$$

$$\boldsymbol{P}(\text{ring}) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{pmatrix} 0 & 1 & 2 & 3 & 2 & 1 \\ 1 & 0 & 1 & 2 & 3 & 2 \\ 2 & 1 & 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 0 & 1 & 2 \\ 2 & 3 & 2 & 1 & 0 & 1 \\ 1 & 2 & 3 & 2 & 1 & 0 \end{pmatrix} \end{array}$$

我们通过将路径矩阵的非零元素加起来除以非零元素个数来计算平均路径长度。如果首先计算行的和，它们的模式将引出一个可以应用到一般情况下的结果。例如，线形图矩阵的所有非零元素的总和等于70，环形图矩阵的所有非零元素的总和等于54。在每种情况下，非零元素的数量为 $n(n-1) = 6(5) = 30$，因此平均路径长度为

$$\text{avg\_path\_length(line)} = \frac{70}{30} = 2.33\ ,\ \text{avg\_path\_length(ring)} = \frac{54}{30} = 1.8$$

n 个节点的线形图和环形图的平均路径长度是多少？运用下列策略推导出答案：

1. 对于偶数和奇数两种情况，分别分析路径矩阵。

2. 获得一个将路径矩阵的所有非零元素加起来的公式，称之为总和 T。在线形图的情况下，T 为非对角元素之和。在环形图的情况下，T 为路径矩阵行的和。

3. 为 T 总结出公式，并注意对称路径矩阵的非零元素的数量等于 $n(n-1)$。

4. 平均路径长度为 $T/n(n-1)$

5. 这种策略生成

$$\text{avg\_path\_length(line)} \sim O\left(\frac{n}{3}\right)\ ,\ \text{avg\_path\_length(ring)} \sim O\left(\frac{n}{4}\right)$$

根据这种策略，每类图的平均路径长度可以通过将所有非零元素加起来然后除以 $n(n-1)$（即非零元素的数量）计算而得。（n 个对角元素为零，如此一来余下的 $n(n-1)$ 个元素非零。）因此，我们必须找到一个对所有非零元素求和的通用公式，并除以 $n(n-1)$，就得到了平均路径长度。

首先考虑线形图。从 $n = 6$ 的路径矩阵开始，注意有多少个非对角元素等于1，以及有多少个等于2、3、4和5。矩阵是对角的，因此就有 $2(n-1)$ 个元素等于1，$2(n-2)$ 个元素等于2，$2(n-3)$ 个元素等于3，$2(n-4)$ 个元素等于4，$2(1)$ 个元素等于5。一般来讲，有 $2(n-i)$ 个元素等于 i，i 是从1到 $(n-1)$，因此我们通过从1到 $(n-1)$ 的 i 将 $2i(n-i)$ 加起来，得到总的 T：

$$\text{矩阵总和} = T = 2\sum_{i=1}^{n-1}\{i(n-i)\} = 2\left[n\sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} i^2\right]$$

假定 $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ 及 $\sum_{i=1}^{n-1} i^2 = \frac{(n(n-1)(2n-1))}{6}$，那么

$$T = n^2(n-1) - \frac{n(n-1)(2n-1)}{3} = n(n-1)\left[\frac{n-(2n-1)}{3}\right] = \frac{n(n-1)[n+1]}{3}$$

$$\text{avg\_path\_length} = \frac{T}{(n(n-1))} = \frac{(n+1)}{3} \text{ 或者 } O\left(\frac{n}{3}\right);\ n \gg 1\text{（线性）}$$

接下来将注意力转移到环形图。矩阵 $\boldsymbol{P}(\text{ring})$ 不同于 $\boldsymbol{P}(\text{line})$，因为 $\boldsymbol{P}(\text{ring})$ 的每行元素从1上升到（$n/2$）然后再下降回到1。所有各行加起来具有相同总和，因为它们包含相同的上升和下降序列。进一步来讲，当 n 为偶数时上升到的最大值为（$n/2$），当 n 为奇数时上升到的最大值为（$n-1)/2$。例如当 $n = 6$，行1升序1，2，3然后为下降序2，1。行2进行同样的工作：1，2，3，2，1。行3进行同样的工作：1，2，3，2，1，……这就意味着每一行的总和是相

同的，即9。相似地，对于$n=7$，每一行是相同的：1，2，3，3，2，1。但是要注意对于奇数n的上升和下降序列是对称的。3出现了两次，因此行总和等于12，或者$2\sum_{i=1}^{3} i = 2(4(3)/2) = 12$。一般来讲

$$\text{row\_total} = 2\sum_{i=1}^{(n/2)-1} i + \left(\frac{n}{2}\right); \text{偶数 } n$$

$$\text{row\_total} = 2\sum_{i=1}^{(n-1)/2} i; \text{奇数 } n$$

有n行，因此$T = n(\text{row_total})$。但是平均路径长度为$T/(n(n-1))$，因此

$$\text{avg\_path\_length} = \frac{(\text{n(row\_total)})}{(n(n-1))} = \frac{\text{row\_total}}{(n-1)}$$

$$= \frac{(n/2)^2}{(n-1)} = \frac{n^2}{(4(n-1))}; \text{偶数 } n$$

$$\text{avg\_path\_length} = \frac{(n+1)}{4}; \text{奇数 } n$$

假定$n \gg 1$，以便$n/(n-1) \sim 1$，环的平均路径长度为$\text{avg_path_length} \sim O\left(\frac{n}{4}\right); n \gg 1$ [环形]

总之，线形图的平均路径长度接近于线形图规模n的三分之一，环形图的平均路径长度接近于环形图规模n的四分之一。

2.5.2 结构化图与随机图

图2-5d的环形图的邻接矩阵和拉普拉斯矩阵几乎是三角形的，如下面所示。在$\boldsymbol{A}$和$\boldsymbol{L}$中的大部分元素为零，因为环形图是*稀疏的*，意味着与最大可能数相比它具有相对较少的链路数。但是，两个非对角包含一个1，对应于连接后续节点的链路链。三角属性并不十分准确，因为1在右上角和左下角。这些是“围绕”链路：

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\boldsymbol{L} = \begin{pmatrix} -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{pmatrix}$$

拉普拉斯矩阵与邻接矩阵相似，将（-2）添加到其对角的每一个元素。如此，在环形网络中的所有节点的度等于2。拉普拉斯矩阵既包含邻接信息，也包括度信息。此外，其行和列加起来等于零。在后面的章节中当我们分析网络的谱属性时，这将成为一个重要的特征。

图2-5e中的图是随机的，因为链路是随机地放在节点对之间。随机图的构造是随机选择尾部，然后随机选择一个头节点，再用链路连接起来。对应于图2-5e的映射函数使用随机数 r_t 和 r_h 选择节点。随机数 r 是从在［0，1）区间的均匀分布上取样获得——也就是使用计算机中的随机数生成器。因此，将 r 放大 n 倍就产生一个从0到 $n-1$ 的节点数，增1会将节点数移动到区间［1，n］中：

$$f_{\text{random}} = [e_i : v_{1+r_t n} \sim v_{1+r_h n}]; \quad i = 1,2,\cdots,m, \text{其中 } m \text{ 为链路数}$$

随机图的邻接矩阵和拉普拉斯矩阵将包含随机放置的1和0，就是因为没有模式。如果图的映射函数建立某种模式——可见或在邻接矩阵，我们将称它是*结构化的或规则的*。如果没有可识别的模式出现，我们就说图是*非结构的或随机的*。因此，一个极端是结构化的或规则图类，另外一个极端是非结构化的或随机的图类。在下一节中我们介绍在规则图和随机图之间还有其他几类图。

2.5.3 k-规则图

图2-5d的环形图是一个非常简单的规则图例子。事实上，它是一个1-规则图，因为每个节点有一个后继节点。我们通过将每一个节点链接到它的直接后继建立环形图，并且由于每个节点都是这样实现的，环“围绕”以至于每个节点的前继也与它链接。

图2-6显示 k-规则图（k = 2,3和（$n-1$））的额外例子。在图2-6b中，每个节点链接到两个后继——后继（$i+1$）和后继（$i+2$）。因此，每个节点具有度 $2k=4$。进一步讲，k-规则图的平均路径长度减少到 $1/k$，因为图分解成如下可达的 k 段：

1. 初始沿着每次跳过 k 个节点的远距离链路，直到到达“相邻的”所需的终止节点为止。
2. 接下来，沿着（$k-1$），（$k-2$），…中等距离链路缩短邻接点。
3. 最后，沿着单跳链路到达终止节点。

例如，图2-6d显示一个带有 $n=10$ 个节点的3-规则图。按照上述设计的公式，环形（1-规则）图的平均路径长度为10/4 = 2.5。一个3-规则图将路径长度缩减大约1/2。因此，这种3-规则图内的平均路径长度大约为(2.5/2) = 1.25（准确答案为120/90 = 1.33）。当我们在下一章中研究自组织网络的特性时，这个发现会非常有用。但是这足以证明，增加 k 会降低平均路径长度。

图2-6c是一个完全图，因为每个节点都与其他节点相连。完全图是密集图的一个例子，因为它包含了可能没有循环或重复链路的最大链路数。

完全图具有 $m = n((n-1)/2)$ 条链路，因为第一个节点连接到（$n-1$）条其他节点上，第二个节点连接到（$n-2$）个其他节点上，依此类推，直到最后一个节点连接到最后剩余的一个节点上。因而链路的总数等于（$n-1$）个整数和：

$$m = (n-1) + (n-2) + \cdots + 1 = \sum_{i=1}^{(n-1)} i = n\frac{(n-1)}{2}$$

k-规则图一般包含 kn 条链路，如果 n 是偶数，$k = 1,2,\cdots,(n/2)$；如果 n 是奇数，则 $k = 1,2,\cdots,(n-1)/2$；k-规则图一般包含 k 个环形，每个环具有 n 条链路，但有不同的后继节点。映射函数链接后继（$i+1$）（对于 $k=1$）、（$i+2$）（对于 $k=2$）等，但是仍旧有 n 条链路才会到达完全图的极限。总的说来，k-规则图包含 $m = \text{minimum}\left(kn, n\frac{(n-1)}{2}\right)$ 条链路，节点度为 $2k$。

2.5.4 图密度

k - 规则图为图理论学家提供了进一步研究的素材，因为它们是稀疏图和密集图之间的桥梁。k - 规则图对于小的 k 来讲是稀疏的，对于大的 k 则是密集的。让图密度近似于图 G 中的链路数与完全图中链路数之比：

$$\text{Density}(G) = \frac{\text{链路数}}{\left(n\dfrac{(n-1)}{2}\right)} = \frac{2m}{(n(n-1))}$$

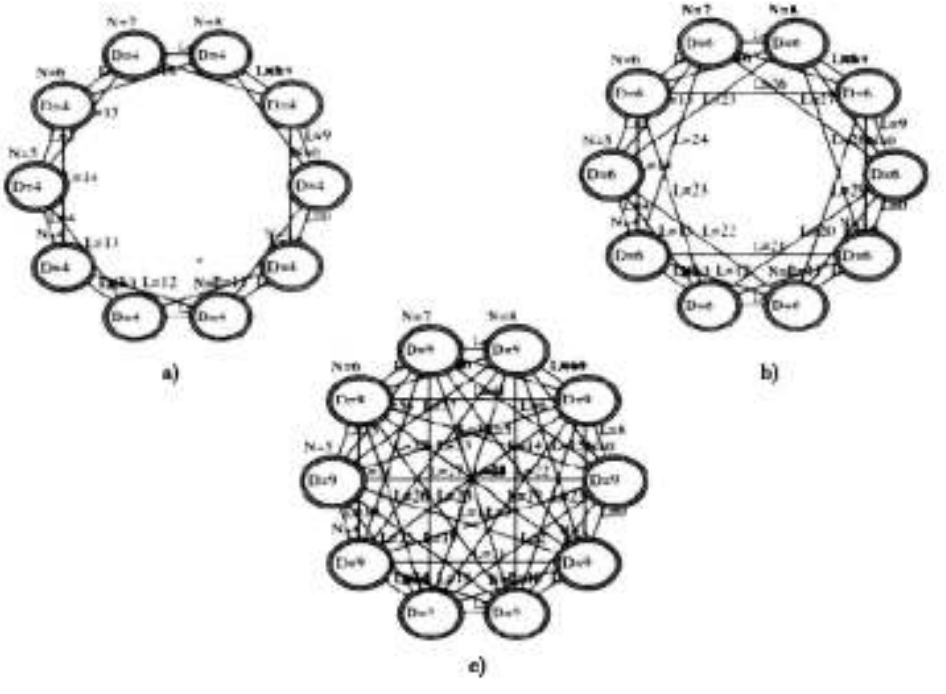


图 2-6 k - 规则图：a）2 - 规则图节点连接两个连续的后继；b）3 - 规则图节点连接三个连续的后继；c）完全图将每个节点链接到每一个其他节点

对于 k - 规则图，给定 $m = kn$，k - 规则图的密度从小到大按照

$$\text{Density}(k\text{ - 规则}) = \frac{2kn}{(n(n-1))} = \frac{(2k)}{(n-1)},\ k = 1,2,\cdots,\frac{n}{2}\ (\text{偶数})$$

变化，根据这种度量，k - 规则图密度是一个 k 的线性函数。在一般图 G 中，密度按 m 链路数线性地增加。可能的链路数按照 $(n(n-1))/2$ 增加，这就意味着大小为 n 的图的可能的链路数增加要比实际链路的增加快得多。这种非线性效应在后面章节中将有重大的影响，因为它允许稀疏图以相对少量的链路展示惊人的属性，例如自组织、自同步和其他形式的涌现。这也是在第 5 章中详细研究的小世界效应的根本原因。

2.6 拓扑结构

在本节中，我们探索各种图的属性以及量化这些属性的度量。具体来说，我们按度序列分布和熵定义图的结构。直观来讲，图的结构受限于其拓扑。如果映射函数 f 一致地将链路映射到节点，那么图就是结构化的。如果映射是随机的，或者缺少模式，图就是随机的。但是这是一个直观的定义。我们想要找到拓扑以及任意图的随机性的更正式的定义。度序列分布为按拓扑对图进行分类提供了一种机制，熵提供一种对随机性的测量。

2.6.1 度序列

令 $g = [d_1, d_2, \cdots, d_n]$ 定义一个度序列，包含 G 中所有 n 个节点的度值。g 中元素的顺序并不重要，但是按以下方式进行列举将会很方便：元素 1 对应着节点 1 的度，元素 2 对应着节点 2 的度等。例如，图 2-4a 的度序列为 $g = [5,3,3,3]$。

g 的元素不必是唯一的，也不必遵守某种顺序。的确，g 内包含的信息并不能保证 G 可以唯一地从 g 中重构出来。但是如果它能，我们就讲 g 为图解的。$g = [5,3,3,3]$ 的一个图解表示显示在图 2-4a 中，但是读者可以构建几个具有 $n = 4$ 个节点和度序列 $g = [5,3,3,3]$ 的图。总的来讲，g 是包含在 G 中的有损信息压缩，可以是图解的，也可以不是。

我们可以进一步压缩包含在 G 中的结构化信息，方式为构造度序列分布 $g' = [h_1, h_2, \cdots, h_{max\_d}]$ 如下：

$$h_1 = \text{度为 1 的部分节点}$$
$$h_2 = \text{度为 2 的部分节点}$$
$$\cdots$$
$$h_{max\_d} = \text{度为 max\_d}(G \text{ 的最大度(hub)}) \text{ 的部分节点}$$

例如，g 包含零个度为 1 的节点，零个度为 2 的节点，三个度为 3 的节点，零个度为 4 的节点，一个度为 max_ d = 5 的节点。因此，图 2-4a 的度序列分布为

$$g' = \left[0,0,\frac{3}{4},0,\frac{1}{4}\right] = [0,0,0.75,0,0.25]$$

g' 的元素总和为 1.0，很像一个概率密度函数。事实上，在某些应用中，将 g' 的元素表示成概率会很有益的，建议新链路连接到节点 u 的概率等于具有度 $d(u)$ 的部分节点。这种思想导致如图 2-6 所示的图的度序列柱状图。

2.6.2 图的熵

图 G 的熵，也就是 $I(G)$，是对有无图结构的一种测量。熵是从信息论中借用的一个术语。它是对“信息量”或信息传递的“不确定”的度量。信息的基本单位是比特，因此熵是图中“随机性”的比特数。熵越高，图随机性就越高。

更正式地讲，图 G 中的“随机性”是其度序列分布 g' 的熵。熵的测量单位是比特，因此设 $I(G)$ 为 g' 中期望的比特数，如下：

$$I(G) = -\sum_{i=1}^{max\_d} h_i(\log_2(h_i)), \text{其中 } g' = [h_1, h_2, \cdots, h_{max\_d}]$$

该方程的含义是什么？考虑一种简单的抛硬币实验，这里抛正面的概率是 $P_{head} = \frac{1}{2}$，抛背面的概率是 $P_{tail} = \frac{1}{2}$。每次丢硬币的结果包含一比特位信息，因为：

$$I(\text{均匀硬币}) = -((P_{head})(\log_2(P_{head})) + (P_{tail})(\log_2(P_{tail})))$$
$$= \left(-\left(\frac{1}{2}\right)\left(\log_2\left(\frac{1}{2}\right)\right) - \left(\frac{1}{2}\right)\left(\log_2\left(\frac{1}{2}\right)\right)\right) = 1.0$$

如果硬币是不均匀的，情况又如何？假设硬币总是倾向于朝向正面，例如，$P_{head} = \frac{3}{4}$，$P_{tail} = \frac{1}{4}$。在这种情况下，$I(\text{不均匀硬币}) = 0.811$。接下来，每次抛硬币包含部分一个比特的信息。每次抛的结果不出意外，因为正面是背面的 3 倍。随着抛硬币更加具有倾向性，结果就更加确定，因此抛硬币具有较少的熵。具有倾向性的硬币比起均匀的硬币具有较小的熵，因为均匀的硬币“更加随机”。事实上，在简单的抛硬币实验中最大的熵值为 1 比特，最小可能的熵值为零。

因此，抛硬币最大的随机性发生在熵为1.0时，最小的熵发生在硬币特别不均匀的时候，即硬币总是朝向正面（或者背面，当不均匀性倾向于背面时）。

熵是对随机性的测量。图的熵越高，就越随机。随着熵降为零，对应图的随机性也降为零。如果将熵计算应用到图的度序列中，就得到图的随机性测量。例如，考虑由以下给出的度序列的两个图：

$$G_1 : g'_1 = \left[0, \frac{1}{4}, \frac{1}{8}, \frac{1}{2}, \frac{1}{8}\right], \quad G_2 : g'_2 = \left[0, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right]$$

$$I(G_1) = -\left(0.25\left(\log_2\left(\frac{1}{4}\right)\right) + 0.125\left(\log_2\left(\frac{1}{8}\right)\right) + 0.5\left(\log_2\left(\frac{1}{2}\right)\right) + 0.125\left(\log_2\left(\frac{1}{8}\right)\right)\right)$$

$$= -(0.25(-2) + 0.125(-3) + 0.5(-1) + 0.125(-3)) = \frac{(4+3+4+3)}{8} = \frac{14}{8} = 1.75 \text{ 比特}$$

$$I(G_2) = -4(0.25(-2)) = \frac{8}{4} = 2 \text{ 比特}$$

在该例子中，G_2 要比 G_1 更随机，因为2比特要比1.75比特大。考虑这两种图的区别的另外一种方法是，G_2 的度分布序列要比 G_1 更具*不确定性*，因为度序列 G_2 包含更大的可变性。这样，G_1 的结构就更加不确定。

随着图随机性的降低，其熵 $I(G)$ 也在降低。因此，我们期望规则图包含零熵或非常小的熵。例如，若 $n = 100$，$m = 200$，完全图或 $(n-1)$ -规则图的熵为零（无随机性）；星形图的熵为0.0807比特（除了hub之外所有节点具有度1）；线形图为0.1414比特（起始和终止节点的度与所有其他节点的度不同）；随机图为2.9比特（链路的分布为“随机的”）。因此，这些例子的随机性是从 k -规则图的0增加到随机图的2.9比特。线形图几乎是星形图随机性的两倍，是因为线形图中节点中的度变化大于星形图中的度变化。

度序列分布告诉人们有关图的“形状”的信息，熵则告诉大家图的“形状”是否有规则性的信息。图的形状越有规则，随机性就越小，因此图中熵就越小。随着深入阅读本书，就会越明白地了解在规则性和熵之间存在一种按比例增减的关系。一个极端是规则图具有很小的熵，另一个极端是随机图具有很小的规则性。

2.6.3 无标度拓扑

图2-7a显示了一个由随机性选择节点构造图的柱状图。这种图的一般形状近似于*泊松分布*，因为节点对的随机选择是一个*泊松过程*——在 m 次尝试中刚好获得 k 次成功的概率，由二项式分布给出：

$$B(k,m) = C\binom{m}{k} p^k (1-p)^{m-k}$$

$B(k,m)$ 由泊松分布近似，在 $B(k,m)$ 中通过将 p 替换成 (λ/m)，并让 m 无限地增长，可得：

$$H(k) = \lambda^k \frac{exp(-\lambda)}{k!}, \text{ 其中 } \lambda = \text{平均节点度}; k = \text{节点度}$$

分布 $H(k)$ 如图2-7a中所示，这里 k = 节点度（水平轴，横坐标），$H(k)$ 则是具有度为 k 的节点（垂直轴，纵坐标）的概率。换句话来讲，$H(k)$ 是具有度为 k 的部分节点。在该公式中，注意 $k!$ 是阶乘函数，$k(k-1)(k-2),\cdots,(1)$。该公式将在第4章进一步讨论。

图2-7b显示了度分布接近于*幂律分布* $H(k) = k^{-q}$（对于 $q > 1$）的柱状图。这种图被称为*无标度的*，因为幂律分布没有缩放标度的参数；也就是说，同有限变量的大多数其他分布（如高斯或正态分布）相比，幂律变量是无限的。幂律分布有时又称为“厚尾分布”，因为随着 k 的增加不会像指数或正态分布那样快地减少。

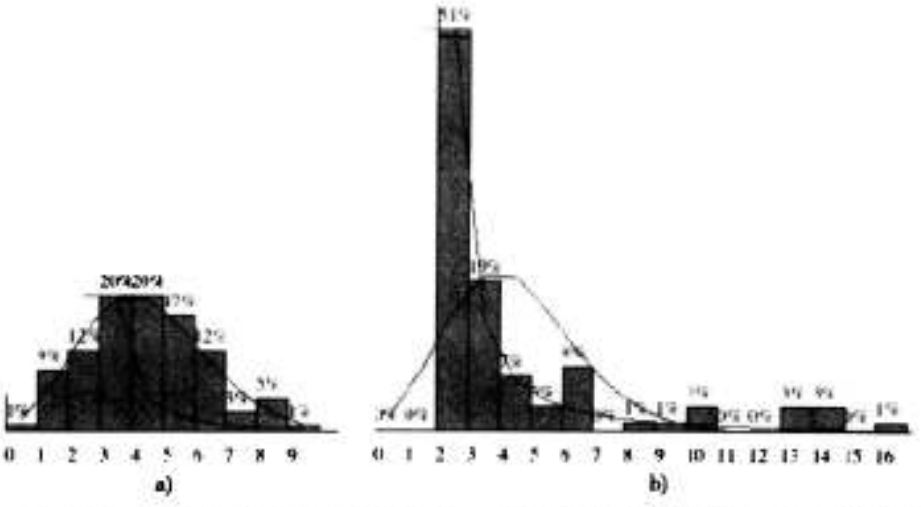


图2-7 度柱状图：a）随机图；b）无标度图。一个线形图显示泊松分布，另一个线形图显示幂律分布拟合柱状图数据

这是一种对图2-7a含义的相当直接的分析。随机图或多或少均匀地跨越 n 个节点分布链路。但是如何解释图2-7a的含义？非常简单，无标度图是畸形的——大量的节点仅有一两条链路，而很少节点会有上百条链路。实际上，大多数高度连通的节点为图的hub，它具有最高的度。无标度图的熵一般要低于随机图的熵，因为无标度网络包含某些结构。例如，对于 $n = 100$，$m = 200$ 的网络来讲，无标度网络具有2.3比特的熵，随机网络具有接近2.9比特的熵。这些估计是通过程序 Network.jar 经过平均 Network.jar 的熵计算的五个样本获得的。

无标度网络比起结构化的极端来讲更接近于随机“图结构”谱的尾部。在一端是结构化 k-规则图（熵等于零），另外一端是不可识别的结构或拓扑的随机网络类型。之间则是具有某些结构的无标度网络类型，因为度序列分布服从幂律分布。

小世界网络刚好相反——稍稍有点结构化和部分随机的网络，更加靠近结构化的谱的尾部（参见表2-3中的对比）。

2.6.4 小世界拓扑

小世界图 G（小世界）是具有相对较小的平均路径长度、相对较高的聚类系数 $CC(G)$ 的图。总的来讲，小世界平均路径长度呈 $O(\log(n)/\log(\lambda))$ 增长，聚类系数倾向于大于50%。什么是聚类，我们如何计算其系数？

假设我们将连接一组 k 邻接节点的链路数与由一个具有 $(k(k-1))/2$ 条链路的完全子图给定的链路最大可能的数相比。如果是完全的，每个子图围绕一个节点，其中 k 节点邻居形成一个三角子图，如果失去一条或多条链路就不构成三角子图。聚类量要根据在这些邻居中找到的三角数量来定。设节点 u 的邻居直接连接到节点 u。存在 degree(u) 个邻居，这就意味着在这些邻居中可能会有 (degree(u)[degree(u) − 1])/2 条链路。假设邻居共享 c 条链路，那么节点 u 的聚类系数 $Cc(u)$ 为：

$$Cc(u) = \frac{2c}{\text{degree}(u)(\text{degree}(u) - 1)}$$

一种更加明显的考虑聚类系数的方式如下。一个具有 degree(u) 个邻居的节点 u 可能是三角子图 c 的一部分，每个都包含节点 u 作为三角的三个节点之一。聚类系数是三角子图 c 的实际数与最大可能数之比。

图2-8演示了为不断增加的更大系数的三个聚类的计算。在所有例子中，degree(v_i) = 4，

因此分母为12。在图2-8a的星形图中，节点v_1的聚类系数为零，因为它的邻居节点没有相互连接；也就是说，v_1不是三角子图的一部分。

接下来在图2-8b中查看围绕节点v_1的拓扑。在图2-8b的星－环组合图中，节点v_1的聚类系数为$Cc(v_1) = \frac{2(4)}{12} = \frac{8}{12} = \frac{2}{3}$，因为$v_1$的4个邻居之间有4条共享链路。还有4个三角子图包含节点v_1。

类似地，在图2-8c中的系数$Cc(v_1)$为(2(6))/12 = 1.0，因为在节点v_1的4个邻居中有6条链路。也有6个三角子图包含节点v_1。聚类是一个包含一个或多个三角子图的子图。

在图2-8中，每个节点带有附属的聚类系数。例如，在图2-8b的星形图中的节点v_2的聚类系数为(2(2))/(3(2)) = 1/3，因为节点degree(v_2) = 3，并且在邻居中有两条链路。图2-8c中的所有节点具有等于1.0的聚类系数，因为图2-8c为一个完全图。

整张图G的聚类系数就是所有节点系数的平均：

$$CC(G) = \sum_{i=1}^{n} \frac{Cc(v_i)}{n}$$

例如，在图2-8b中，星－环组合图中，所有节点的聚类系数为2/3，因此整个图的聚类系数为

$$CC(G(\text{星-环图})) = \frac{\left(5\left(\frac{2}{3}\right)\right)}{5} = \frac{2}{3}，\text{或者 } 0.67$$

小世界图由一个比随机图大得多的聚类系数所描述，但是也有某些属性是和随机图所共有的。例如，小世界图和随机图具有相似的度分布和直径。

表2-2列出了一些常见的小世界网络图和对等的随机图。表2-3比较了随机图、无标度图、小世界图和2－规则图的属性。小世界的聚类系数可以比对等同样大小和对等链路数的随机图大几个数量级。

表2-2包含了许多将会在随后的章中详细探讨的例子。万维网图的构建是将文档表示成节点，对文档的索引表示成链路。互联网图的构造是将自治系统（互联网服务提供商）表示成节点，网络连接表示成链路。相似地，好莱坞电影演员表示成节点，电影中合作关系由链路来表示。发表科学论文的合作者由节点来表示，之间的链路表示他们合作了一篇论文。电厂、变电站以及美国西部电网的配电网建模成节点，而输电线建模为链路。在秀丽小杆线虫（C. elegans）神经网络中，神经元是节点，它们的连接为链路。最后，食物网络通过由猎物（节点）和捕食者（链路）建模成的某种食物链构成。

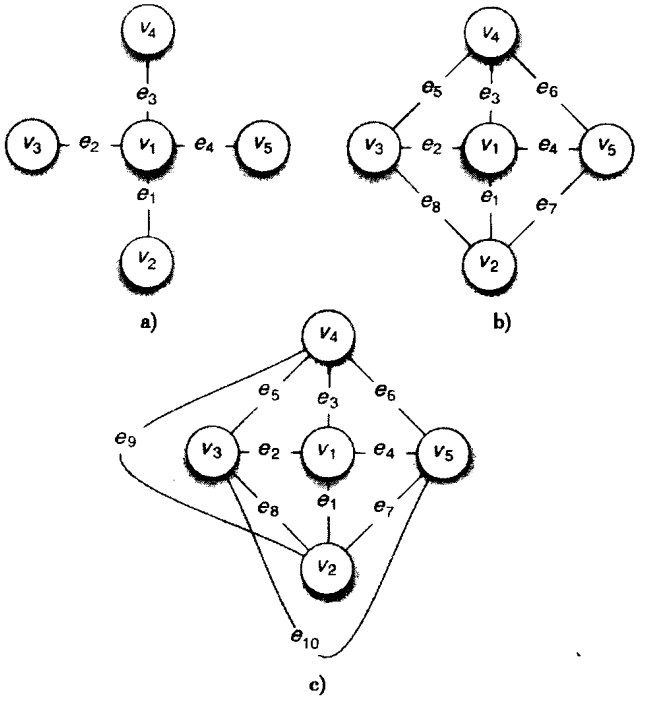


图2-8 聚类系数计算：a）简单星形图（$C_v = 0$）；b）星－环图（$C_v = \frac{2}{3}$）；c）完全图（$C_v = 1.0$）

本书附带的程序 Network. jar 可以用来进行如下实验。假设 $n = 100$ 和 $m = 200$。使用 Network. jar（或 Network. html）产生一个 2 - 规则图、无标度图、小世界图和随机图，然后记录每个属性如表 2-3 中所示。这些图有哪些一般性的规律呢？

表 2-2 小世界网络的一些公共例子

图	大小，n	小世界聚类系数	随机聚类系数
万维网	153 127	0. 11	0. 00023
互联网	6 209	0. 30	0. 00100
同一电影中的演员	225 226	0. 79	0. 00027
科学论文的合作者	52 909	0. 43	0. 00018
美国西部电网	4 941	0. 08	0. 00200
秀丽小杆线虫神经网络	282	0. 28	0. 05000
食物链（生态链）	134	0. 22	0. 06000

表 2-3 图（n=100，m=200）的某些属性对比，对于小世界图 p=5%

属性	随机	无标度	小世界	2 - 规则
hub 度	10	21	10	4
平均度	4. 0	3. 94	4. 0	4
分布	泊松分布	幂律分布	类似泊松分布	Delta（4）
平均路径长度	3. 38	3. 08	4. 0	12. 88
直径	7	5	9	25
聚类系数	0. 045	0. 156	0. 544	0. 500
熵	2. 9	2. 3	0. 9	0. 0

表 2-3 为图结构总结，包括从随机图到纯的结构化 2 - 规则图。我们可以从分布类型、聚类系数的相对值以及随着结构的增加而下降的熵加以归纳。例如，“最随机的”的图的度序列分布是泊松分布，但是随着随机性等级的下降（在表中从左到右），分布变得不那么像泊松分布了。随着随机性等级的下降，每类图的熵下降。只有一个例外（小世界），图越随机，聚类系数就越低。

注意减少熵值——无标度图要比随机图具有更低的熵，但是要比小世界图或 2 - 规则图的高。换句话来讲，无标度图要比结构化的图更随机。反过来，小世界图要比随机图更加结构化，因为它的熵要低于所有其他类，除了 2 - 规则类之外。表 2-3 支持以下推测，无标度图和随机图要比小世界图和规则图“更随机”。我们将在后面的几章中探索这种推测。

随机图类具有最小的聚类系数，意味着较少的结构。小世界图具有最大的聚类系数，即使它的度序列分布非常类似于随机图的。不寻常的高聚类是小世界图的一个突出特点。

无标度图有很多度非常低的节点和一个度非常高的节点。因此，其度分布偏向左——朝向较小度的值。因为这么多的节点连接到图的 hub 上，无标度图也显示了较小的图直径——这与随机图相似。它的直径接近于小网络图直径的一半，50% 要比随机图的直径小。这是高度集中的链路附加到 hub 上的结果。出乎意料大的 hub 度是无标度图的一个突出属性。

所有图中“最结构化”的图是 2 - 规则图，它具有非常高的平均路径长度和高聚类系数。这很显然，因为 k - 规则图具有很多短链路，因此，它要经过很多跳才能到达图的较远处。短的链路也形成具有相对高聚类系数的邻居。

在表 2-3 中，每类图的熵从 0 比特增加到 2. 9 比特。这表示在随机图的拓扑中具有大量的不

确定性，在2－规则图中具有大量的确定性。无标度图更像随机图而非小世界图，小世界图则更像规则图。结构层次化存在于很多类图中：规则图、小世界图、无标度图和随机图。规则图和小世界图倾向于由包含一些随机性的小世界图构建。无标度图和随机图倾向于由包含一些结构化的无标度图的非结构化。

表2-3中显示的小世界图的结果倾向于规则结构。详细地讲，小世界图已经调整过，仅有5%的链路是随机的，而95%是规则的。这是由表2-3中设置的“$p = 5\%$”所标示的。在下一章中，我们阐述如何调整小世界图，通过将一部分链路从规则连接转换成随机连接，这样随机化一部分映射。

调整将导致部分结构化以及部分随机的小世界。调整也更改小世界类图中的熵——从零到随机网络的100%。当可调整参数设置为零条随机链路时，小世界图具有零熵，当设置为100%随机链路时，小世界图具有和随机图相同的熵。因此，小世界图可以调整为随机和2－规则结构之间结构层次中的任何位置。表2-3仅显示了一种可能。

2.7 软件中的图实现

研究图的最好方法是在运行中加以观察。幸好，我们有足够的廉价的快速计算机，可以相当轻松地处理大的结构，例如图。的确，本书提供的计算机程序为例子值的网络处理软件。我们不仅验证本书描述的理论结果，而且要进行大量的数学实验以便深入了解。软件使得获取网络科学的经验数据以及理论经验成为可能。

Java编程语言用来构建软件模块，可以集成到读者自己的软件之中，或简单地用来理解这里描述的程序如何工作。选择Java是因为它几乎可以在任意计算机上工作，并且它与其他常用程序语言如C++相类似。它也很容易阅读和理解，因此可以说它是有些自文档化的。

2.7.1 Java节点和链路

图在计算机中可以以多种形式表示。例如，我们可以使用矩阵代数并在两维数组中存储G的节点和链路。我们可以使用链表数据结构——节点为数据类型并且链路为指针。或许最容易表示为类结构和简单的数组结构。这种方法既有优点也有缺点。主要的优点在于简洁并且与关系数据结构兼容。缺点是为了简洁要牺牲掉在处理上的花费。这里使用的简单矢量方法需要大量的计算——有时进行一些简单的操作，但是之所以使用是因为其简洁和容易理解。

假设节点和链路被计算机序列化的类表示。在Java中的类是一种抽象，定义存储区域以及允许在存储上进行的操作方法。类被实体化成对象——在计算机内存中的实际存储。序列化类定义成可以以单个对象保存到磁盘上的对象。在Java中，对象是自动地序列化的，以便于使用单个读/写命令来完成到和从磁盘的输入输出。

Link类用名字、尾部、头部、值、颜色和选择标记定义图链路，以便指示用户何时通过点击鼠标选择链路。Link的头部和尾部是Link连接的节点对。因此，一条Link映射到一节点对就像定义$G(N,L,f)$所描述的那样。在Java中，Link类实现链路及其属性：

```
class Link implements java.io.Serializable {
    public String name = "";                    //Name of link
    public int tail;                            //From Node #
    public int head;                            //To Node #
    public double value = 0;                    //Value of link
    public Color c;                             //Link Color
    public boolean selected = false;            //Mouse down selection
}//Link
```

一个图的节点就是类Node的一个实例，它也是序列化的，因此它可以存储在磁盘上。这种

类包含了不会因为从一个实例变为另外一个实例而变化的静态变量。例如，对于所有节点，屏幕的显示尺寸是相同的。此外，每个节点有自己的名字、度、颜色、值、半径和屏幕坐标（x，y）。还有，颜色代码 int_color 是必需的，因为 Java 不序列化 Color，因此 int_color 是一个整数编码类型的 Color。

```
class Node implements java.io.Serializable {
    //Static values
    public static int xstep = 40;          //Minimum size of display node
    public static int ystep = 30;          //Dynamic variables
    public String name = "";               //Name of node
    public int degree = 0;                 //Degree of node
    public int out_degree = 0;             //Out Degree = # directed
                                             links out
    public int in_degree = 0;              //In Degree = # directed links
                                             in to node
    public Color color = Color.white;      //Red = infected or source;
                                             green = sink
    public int int_color = 0;              //Hack to defeat Java
                                             serializer limitation
    public int x=0;                        //Location on screen
    public int y=0;
    public double next_state = 0;          //Next State used to Sync
    public double value;                   //Working value
    public double cluster = 0;             //Cluster coefficient
    public int level = 0;                  //Used to find center
    public int radius = 0;                 //radius of a node
    public boolean visited = false;        //Used for spanning tree
                                             (finding center)
    public boolean center = false;         //True if node is a center
    public boolean outer = false;          //True if node is outer
    public boolean selected = false;       //For mouse down events
    public int timer = 0;                  //Infection timer
}//Node
```

2.7.2 Java 网络

每次当节点或链路创建时，Node 和 Link 就被实例化。我们可以将这些对象链接到一起构成图，而且很容易就可以使图的结构简单化，并让计算机来完成该工作。因此，我们使用简单的数组结构容纳所有的链路，用另外一个简单数组结构容纳所有节点。此外，计算机需要及时记住在任何一点上有多少节点和链路。如此一来，第三个类用来定义图 G。类 theNetwork 构成了图的结构并包括在计算期间需要的许多其他变量。

```
class theNetwork implements java.io.Serializable {   //Class NW_
    public static int maxNodes = 4096;         //Limit of #nodes
    public static int maxLinks = 16384;        //Limit # Links
    public static Node pick = new Node();      //For mouse down event
    public static int pick_index = -1;         //Index of selected node
    public static int prior_selected = -1;     //For adding link between
                                                 selected nodes
    public static int t_max = 5000;            //Max simulation time
    public static int t = 0;                   //Simulated clock
    //Input Network parameters - can be changed by Preferences
    public static int nInputNodes = 10;        //Number of input nodes
    public static int nInputLinks = 20;        //Number input Links
    public static int nConnectionProbability = 100; //Probability of
                                                      rewiring link
```

```
public static double DefaultValue = 1;   //Default node and link
                                            value
public static int SleepTime = 10;         //speed of display thread
public static double PathLength = 0;      //Avg_Path_Length of
                                            network
public static double LinkEfficiency = 0; //Link Efficiency of
                                            network
public static double SpectralGap = 0.0;     //Network Spectral Gap
public static double SpectralRadius = 1.0;  //Network Spectral
                                                  Radius
public static int CentralNode = -1;      //First central node found
public static int ClosestNode = -1;      //First closest node found
public static int ClosestPaths =         //Number of Paths thru
                                            closest
public static double ClosestAvg = 0;     //Average closeness value
public static int Radius = -1;           //Radius of network
public static int Diameter = -1;         //Diameter of network
public static double ClusterCoefficient = -1; //Cluster
                                          Coefficient of the network
public static int HubNode = -1;         //Hub node #
public static int DeathRate = 0;        //Epidemic death rate %
public static int InfectionRate = 20; //Epidemic infection
                                            rate %
public static int RecoveryRate = 20;   //Recovery rate
public static int RecoveryTime = 1;    //SIR time to recover or die
public static double CC = 0.0;         //Cluster coefficient of
                                            network
public static double Length = 0.0;     //Total lengths of all links
public static Matrix M;                //Adjacency and Laplacian
                                            matrices
//GUI parameters
public static int xwide = 840;       //Default Size of drawing area
public static int ytall = 480;       //Default Size of drawing area

    public static int y0 = 240;              //Default Drawing areas
    public static int x0 = 420;              //Default Origin at (x0,y0)
    public static int CircleRadius = 200;   //Default Size of layout
                                               circle
    public static String Message = "";       //User message
    //dynamic data
    public int nNodes = 0;                   //Actual number of nodes
    public int nConsumers = 0;             //Number of consumers - black
    public int nCompetitors = 0;           //Number of competitors - blue
    public int nLinks = 0;                       //Actual number of links
    public Node node[ ] = new Node[ maxNodes] ;  //List of Nodes
    public Link Link[ ] = new Link[ maxLinks] ;  //List of Links
    public boolean doUpdate = true;              //Dirty bit for update
                                                   of displays
    //Constructor
    theNetwork(){    }//theNetwork
}
```

给定了这些类，实例化一个图就会很容易。例如，图 G 是通过声明它的类型 theNetwork 创建的：

```
theNetwork G = new theNetwork();              //Graph G
```

所有用来向图 G 中填充节点和链路的处理方法都包含在类 theNetwork、Link 和 Node 中。theNetwork 方法加前缀 NW_ 将它们指定为 theNetwork 类的成员。例如为了创建一个随机图：

```
public void NW_doCreateRandomNetwork() {
        NW_doCreateNodes();                    //Create nodes, only
        NW_doCreateRandomLinks();              //Create links, randomly
    {//NW_doCreateRandomNetwork
```

NW_doCreateNodes()方法实例化 nNodes 个 Nodes（节点）并将它们存储在数组 node[0..nNodes-1]中，NW_doCreateRandomLinks()方法实例化 nLinks 个 Links 并将它们存储在数组 Link[0..nLinks]中。回顾随机图是由随机地选择节点对所创建的。具体的由 NW_doCreateRandomLinks()的实现提供：

```
private void NW_doCreateRandomLinks() {
    nLinks = 0;                    //Increase links until limit is reached
    int to, from;                      //Randomly selected nodes
    while(nLinks < nInputLinks){   //Don't stop until all links are
                                                    created
        from = (int)(Math.random()*nNodes);
        to = (int)(Math.random()*nNodes);
        while(from == to) to = (int)(Math.random()*nNodes);
        NW_doAddLink(node[from].name, node[to].name);
    }
  }//NW_doCreateRandomLinks
```

节点对（from, to）通过在区间［0, 1）生成的一个随机数随机选择，然后再乘以 nNodes，以此来获得位于区间［0, nNodes-1］的一个随机节点数。但是，这些必须是独特的，以便于方法继续生成一个随机值给予变量 to，直到它与变量 from 不同为止。

调用 NW_doAddLink(from, to)方法将一个新的链路插入到节点对之间，但是如果重复链路已经存在时就返回 false。NW_doAddLink()也会增加 nLinks，这是一个在任意时间点上的链路数。因此，方法继续尝试添加链路，直到 nInputLinks 链路已经被插入到 G 中为止。

将链路插入到节点对之间的多态方法如下所示——一种方法在链路中插入一个名字，而其他节点则不插入。第一种形式假定链路名字就是它们的号数，而第二种形式需要在其参数列表中的一个显式名字：

```
public boolean NW_doAddLink(String from, String to){
     return NW_doAddLink(from, to, Long.toString(nLinks));
 }
 public boolean NW_doAddLink(String from, String to, String name) {
    if( from == to) return false;                  //No self loops
    Link e = new Link();                           //Allocate space
    e.tail = NW_doFindNode(from);                  //Node pair...
    e.head = NW_doFindNode(to);
    e.value = DefaultValue;                        //Assume defaults
    e.name = name;                                 //Link name
    boolean exists = isConnected(e.tail, e.head);     //No duplicates
    if(!exists) {                                 //OK to insert new link
        if(nLinks >= maxLinks){
            Message = "Cannot exceed Maximum Number of Links";
            return false;
        }
        Link[nLinks] = e;                        //Insert in list of links
        Link[nLinks].c = Color.black;            //Make visible
        nLinks++;
        node[e.tail].degree++;                   //Increment degree
        node[e.tail].out_degree++;
        node[e.head].degree++;                   //Both ends
        node[e.head].in_degree++;
        return true;
    }
```

```
        return false;                                   //Duplicate exists
    }//NW_doAddLink
    private boolean isConnected(int na, int nb){
        for(int i = 0; i < nLinks; i++){
            if(Link[ i] .tail == na && Link[ i] .head == nb
                        ||
               Link[i].head == na && Link[i].tail == nb) {
               return true;
            }
        }
        return false;
    }//isConnected
```

练习

2.1 包含 $n = 100$ 个节点的环形图的准确平均路径长度为多少?

2.2 包含 $m = 100$ 条链路的线形图的平均路径长度为多少?

2.3 图 2-4b 的欧拉路径长度是多少?

2.4 构造一个具有 $n = 4$ 个节点以及度序列为 $g = [5,3,3,3]$ 的图。

2.5 重联图 2-4a 以便得到 $e_5:v_2 \sim v_3$。现在找到一个欧拉回路。具体是什么?

2.6 图 2-4b 中，图的（平均）路径长度属性是指什么？说明该图的路径矩阵并从路径矩阵中导出平均路径长度。

2.7 图 2-2a 中，图的度序列 g 是指什么？忽略链路的方向。

2.8 图 2-2a 中，图的度分布是指什么?

2.9 图 2-9 为一个消防局服务的 10 个城区图。该市想要在这些区域的“中间”为一个新的消防站选址。假定在该图中链路代表救火车从邻区到每一个区的转移时间是相同的。该市想要缩短从新的站到所有区的穿行时间，那么新站最好位于哪个区域？图 2-9 中，图的拉普拉斯矩阵是什么？（注意：节点和链路编号为 0 到 9。）该图的映射函数 f 如下：

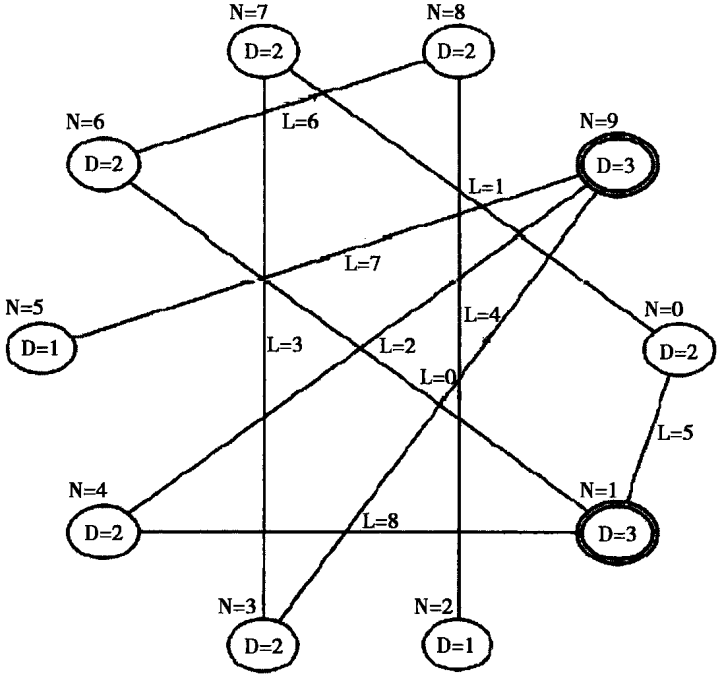


图 2-9 $n = 10$，$m = 10$ 时程序 Network. jar 显示的图 G

$$f = \begin{pmatrix} e_0:v_3 \sim v_9, e_1:v_0 \sim v_7, e_2:v_4 \sim v_9, e_3:v_3 \sim v_7, e_4:v_2 \sim v_8, \\ e_5:v_0 \sim v_1, e_6:v_6 \sim v_8, e_7:v_5 \sim v_9, e_8:v_1 \sim v_4, e_9:v_1 \sim v_6 \end{pmatrix}$$

2.10 在前面的练习中，离中心节点最远的是哪个节点，有多远？

2.11 按照紧度属性，在图2-9中最近的节点是哪个？

2.12 图 G 中，$n = 5$，包含节点 v_1, v_2, v_3, v_4, v_5，具有映射函数 $f = [e_1:v_3 \sim v_4, e_2:v_1 \sim v_2, e_3:v_1 \sim v_3, e_4:v_4 \sim v_5, e_5:v_3 \sim v_5, e_6:v_2 \sim v_5]$。节点 v_5 的聚类系数是多少？

2.13 在练习2.12中，整个图的聚类系数等于多少？

2.14 [高级] 计算图2-4b的简化哥尼斯堡七桥图的谱半径和谱隙。

2.15 [高级] 说明当 n 近似无穷大时，二叉树图近似 $\exp(1) = 2.718, \cdots$ 的谱半径。

第3章
Network Science: Theory and Applications

规则网络

本章根据拓扑定义了一些著名的规则网络，并提供一种构造它们的生成过程。生成过程是一种从一组节点、链路和将链路映射为节点对的规则中生成一个网络的算法。生成过程作为一种计算机方法很容易在面向对象语言如 Java 中实现。我们将给出环形、二叉树、超环形和超立方网络的 Java 方法。

从网络科学的角度来研究规则网络有多种原因：（1）它们可以提供与随机网络的鲜明对比——大多数规则网络具有零或非常小的熵，（2）k - 规则网络是小世界生成过程的基础，（3）规则网络属性在研究任意网络的同步中非常重要——我们将在第 9 章和第 10 章中学习同步。特别是，嵌入到更大网络内的规则子图循环的长度决定着较大网络是否同步。同步对于生物网络的稳定行为、营销和商业稳定以及社会网络中人群达成一致性意见的能力都是必要的。

规则网络也是其他更为复杂网络的基础。在第 5 章中，我们说明如何从 k - 规则网络以重联概率 p 生成一个小世界网络。在这种情况下，初始的 k - 规则网络具有零熵，但是出现的小世界的熵与重联概率成比例。更为重要的是，我们将证明小世界网络的属性是从它们底层的规则网络中继承而来的。例如，Watts-Strogatz（WS）小世界网络从它的最初的 2 - 规则网络中继承了聚类系数。

为了能以平均路径长度和链路数来比较网络的效率，将引进一个新的度量——*链路效率*。一个链路有效的网络将以有效的方式利用它的链路以便减少平均路径长度。在网络科学的许多应用中，我们在保留链路数目的同时，需要尽可能短的平均路径长度。

我们定义链路效率 $E(G)$ 如下：

$$E(G) = \frac{m - \text{avg\_path\_length}(G)}{m} = 1 - \frac{\text{avg\_path\_length}(G)}{m}$$

此外，本章为计算任意网络的直径、中心和平均（特征）路径长度设计了 Java 算法。这些在程序 Network. jar 中实现的算法工具是理解网络科学实验方法的一部分。我们将仿真与数学的严格结合起来，以便推导出下面的平均路径长度和熵等属性。实践数据是从程序 Network. jar 中获取的，分析曲线与数据拟合，然后从推导逻辑中导出通用的直径、中心性和平均路径长度以及链路效率方程。这使得读者对主题有一个直观掌握，同时还要有必要的数学技巧，以便完全掌握主题。

*规则网络*是一种具有规则图结构（链路的重复模式）的网络。由于其规则性，这种类型显示出低熵或零熵。这里研究的网络也显示出链路的经济性，据此每一个节点是以相对较小的跳数从其他节点可达的。规则网络是稀疏的，连通的，并且具有相对较小的直径、小的中心节点半径和小的平均路径长度。这些属性使规则网络成为研究实际网络设计的最佳选择。

我们对具有较小的平均路径长度的稀疏规则网络特别感兴趣，因为它们具有很多应用。例如，在设计多处理器计算机系统时，既要最小化链路数量、平均路径长度，也要最小化将处理器连接到一起的互连网的直径，这一点非常重要。高链路有效网络能够成为很好的通信网络，因为它们能够减少网络延迟和传输延迟。

有效的规则网络也是良好的人类通信结构，因为一个平均路径长度短的组织图会使得公司更加有效率。在快速运转的现代化企业组织中，命令必须尽可能以最有效的方式传输通过稀疏网络。对以网络为中心的组织进行优化，通过减少链路以降低费用，通过减少平均路径长度以便减少延迟。在本章中，我们将分析传统的层次化树形结构网络不如其他类型网络有效。这可以解释为什么现代化组织要比几十年前更少地采用树形结构。

本章我们将介绍以下内容：

1. 为构造二叉树、超环形和超立方网络提供生成过程的 Java 方法。
2. 为找到任意网络的路径长度、直径和中心性属性，设计宽度优先算法及其 Java 实现。
3. 提出一种新的效率（即链路效率）的测量，也就是 1 减去平均路径长度与链路数之比：

$$E(G) = 1 - \frac{\text{avg\_path\_length}(G)}{m}$$

这是一个用来比较二叉树、超环形、超立方网络的度量。

4. 将实验和数学推导方法结合起来应用于求解任意网络的平均路径长度计算问题。程序 Network. jar 生成实际数据，画出与数据拟合的曲线，然后从曲线中推导出平均路径长度公式。我们将在全书中使用这种通用技术。

5. 介绍二叉树网络类型并且证明它们是路径有效率的，因为它们以非常有效的方式利用链路。这类网络的特征路径长度渐近于 $O(\log_2(n))$。链路效率随着网络规模的增加而降低：

$$E = 1 - \frac{2\log_2(n)}{n};\quad n \gg 1$$

6. 介绍了同样具有高水平路径效率的超环形网络——优于二叉树，但是却不如超立方网络有效。这种网络具有 $O(\sqrt{n})$ 的特征路径长度，但是其网络链路效率为

$$E = 1 - \frac{1}{4\sqrt{n}};\quad n \gg 1$$

7. 介绍超立方网络类型，并证明它既是稀疏的又是链路有效的。超立方网络的特征路径长度为 $O(\log_2(n))$，这一点与二叉树的一样，但是它的链路效率为 $E = 1 - (1/(n-1))$。因为超立方网络是这里研究的所有规则网络中链路最有效率的，所以它们是多处理器计算机系统和低延迟通信网络设计中优选的结构。

本章使用程序 Network. jar 生成网络，并分析这里描述的每一种网络。本章中所给实现生成过程的 Java 方法是 Network. jar 中方法的简化版本。

3.1 直径、中心性和平均路径长度

总的来讲，许多实际网络设计的目标是将 n 个节点以最便宜的方式相互连接起来。当网络拓扑具有最少的链路或当穿越网络需要的跳数最小时就是“最便宜的”。在某个应用中，我们可能想要最小化直径（最坏情况下的路径长度），在另外一种应用中，目的可能是最小化平均路径长度（平均情况下）。曼哈顿街布局——南北/东西矩形网格——已经被许多城市采用，以便最小化从一个节点到另一个节点所需的时间。但是一个街道网格需要很多链路——太多以至于不可能有这么多条链路将所有主要城市相互连接起来。因此，采用线形网将大多数城市连接起来，因为这样需要更少的链路。城内街道网格使传输时间最小，但是城际道路网络使得将城市连接起来的链路（道路）数量最小。

在网络平均路径长度上的链路数与跳数的折中，可以通过称为链路效率 E 的度量来获得，链路效率 E 定义如下：

$$E(G) = \frac{m - \text{avg\_path\_length}(G)}{m}$$

这里 m 为 G 中的链路数。注意它的取值范围为从零（当平均路径长度为 m 时）到100%（当平均路径长度为0时）。在极端情况下，$E = 1.0$，每条链路都对链路效率作出了贡献。当 $E = 0$，平均路径长度等于 m，这是可能最差情况下的路径长度。链路效率是一种链路在缩短路径长度下的效率的测量。对应网络的 E 值越高就越有效率。

例如，在第2章中我们设计了线形、超环形和完全网络的平均路径长度估计。将路径长度和链路数的公式插入到链路效率的公式中就得到 E 的一般公式。比较不同网络的 E 是一种确定最有效的链路的方法。表3-1列出了多种网络类型及其链路效率。在下面的章节中，我们导出这些结果并演示超立方网络是规则网络中链路最有效的。按照链路效率，表3-1中将线形和超环形网络列在最低，而超立方、随机和完全网络列在最高。令人惊讶的是随机网络是链路有效的[⊖]。

总的来讲，随着网络规模的增加，我们对链路的有效利用率是否得到提高感兴趣。如果随着网络规模 n 接近无穷大，链路效率接近100%，网络就是可扩展的。否则，网络是不可扩展的。表3-1中的哪个网络是可扩展的？二叉树、超环形、超立方和完全网络显然是可扩展的，因为随着网络规模 n 接近无穷大，链路效率接近100%。但是随机网络如何？答案要取决于链路数 m 是否随着 n 一起扩展。

表3-1 几种网络类型的链路效率，$n \gg 1$

网络类型	效 率	例 子
线形	$\frac{2n-4}{3(n-1)}$	渐近于 $\frac{2}{3}$
环形	$\frac{3n-1}{4n}$	渐近于 $\frac{3}{4}$
二叉树	$1-\frac{2\log_2(n+1)-6}{n-1}$	$n = 127$，$m = 126$，$E = 93.4\%$
超环形	$1-\frac{1}{4\sqrt{n}}$	$n = 100$，$m = 200$，$E = 97.5\%$
随机	98.31%	$n = 100$，$m = 200$，avg_path_length = 3.38
超立方	$1-\frac{1}{n-1}$	$n = 128$，$m = 448$，$E = 99.2\%$
完全	~1.0	$m = n\frac{n-1}{2}$，avg_path_length = 1

路径长度和中心性的计算

链路效率和网络中心性是路径长度的函数，因此我们从设计一个计算任意网络的路径长度的算法开始。一旦我们知道了网络的平均路径长度，我们就能计算直径、中心性和链路效率。这里设计的路径长度算法也可以额外得出直径、中心性和链路效率。

回顾一下对特征路径长度的定义：对于所有从节点 v 开始到节点 w 结束，从节点 v 到节点 w 的所有（最短）直接路径的平均。对于所有的 v 和 w 的组合，路径长度算法必须能够计算沿着 v 和 w 之间的直接路径的跳数。令 t 为路径的总数，$r_{i,j}$ 为节点 v_i 到节点 w_j 之间的直接路径长度。那么所有 $r_{i,j}$ 的平均数为

$$\text{avg\_path\_length} = \sum_i \sum_j \frac{r_{i,j}}{t}$$

⊖ 这是由于小世界效应导致的。

$r_{i,j}$ 是第 2 章中定义的路径矩阵中的元素，$t = n(n-1)$，因为路径矩阵的对角线为零。给定路径矩阵，找到平均路径长度就很容易，计算路径矩阵的元素就等于找到所有路径！我们通过采用直接方法避免循环逻辑——简单地计算所有 n^2 节点对之间的路径长度。理想的路径长度算法能跟踪整个网络，同时小心避免产生回路和较长的替换路径。

这里建议的算法使用宽度优先递归搜索，在连通网络中跟踪从节点 v 到每个其他节点 w 之间的最短距离。这是很费时的，但是除了需要避免回路及替换成（长的）路径之外，实现起来很简单。宽度优先访问直接邻居节点，然后是邻居的邻居等，直到到达一个已经访问过的节点，或到达一个没有其他邻居的节点，返回初始节点（回路）或到达目的地节点为止。

任意网络的宽度优先搜索将围绕它的节点按层组织，如图 3-1 所示。一层等于一跳，因此算法简单地查找围绕起始节点 v 和终止节点 w 的最小的层号。相对于节点 v，终止节点 w 的层号等于从 v 到 w 的跳数。图 3－1 演示了如何使用层号标记邻接的节点以组织宽度优先搜索，而层号等于路径长度。

从路径矩阵来讲，路径矩阵的行 v 对应于起始节点 v，行 v 的元素对应于网络中的其他节点。如果一个节点从节点 v 不可达，那么它们之间的路径长度为零。我们逐行构造路径矩阵，然后计算路径矩阵的非零元素的平均值。从所有节点对的所有路径的平均得到特征（或平均）路径长度。

在每一层可能有多个邻接节点，因此我们采用压栈来记忆当前起始节点 v 的所有邻居。在访问过 k 层的每一个邻居后，我们就将它标记成“访问过”，存储它的层号 k，并将它的所有邻居推入栈。访问并标记 k 层的所有邻居后，我们弹出栈以便获得下一层的节点，重复上述过程。当栈为空时，该过程就停止。

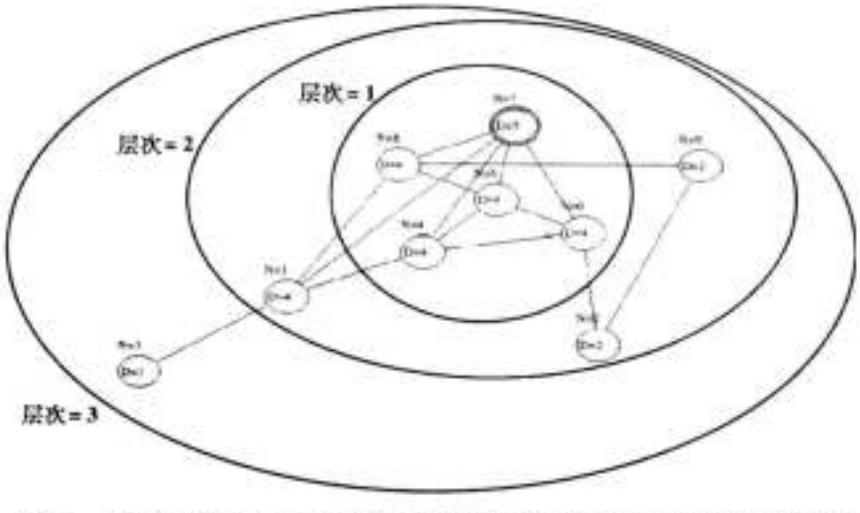


图 3-1 围绕起始节点 $n_0 = v_6$ 按层次布置的任意网络。层 k 表示从 n_0 到邻居和邻居的邻居节点的跳数

此时，让我们计算路径长度，同时还要计算直径和半径。直径是网络中最大的 k 值，节点的半径就是该节点到所有其他节点的最大距离。换句话来讲，节点 v 的半径就是它到离它最远节点的跳数。具有最小半径的节点就是中心节点——它要比任何其他节点离所有节点都近或至少一样近。因此，该过程也产生所有节点中心性的测量，并指定最中心的节点作为网络的中心。中心性是一种等于网络中到所有节点的最小直径的属性。

从网络中查找所有路径时，我们也附带计算了紧度，但是该主题将在第 4 章才研究，因为规则网络中节点的紧度没有多大意义。规则网络非常均匀，对于所有节点来讲紧度大致是相同的，

但是也有例外。例如，线形网络的端节点并不像中间节点那么近。

最后，我们能够计算链路效率，因为我们知道链路数 $m = n$ 条链路，以及网络的平均路径长度。链路效率 E 简化成：

$$E(G) = \frac{1 - \text{avg\_path\_length}(G)}{m}$$

下面 Java 方法 NW_doFindCenters()用来计算直径、中心、平均路径长度，同时计算 E。它实现上述的宽度优先搜索算法，并找到 n^2 节点对之间的所有路径。类 LayoutReply 是一种数据类型，用来将直径、中心节点半径、平均路径长度和链路效率返回给调用程序：

```
class LayoutReply {
  int big;                          //Diameter
  int small;                        //Central radius
  double mean;                      //Average path length
  double efficiency;                //Link efficiency
 }
```

以下 Java 代码是为了简洁性和可读性而非效率书写的。对该代码的改进留给读者作为练习。代码使用压栈（在别处定义），并假设如第 2 章中所定义的节点和链路数据结构：

```
public LayoutReply NW_doFindCenters(){
   LayoutReply reply = new LayoutReply();     //Returned values
   Stack s = new Stack(nNodes+1);
   int k = 0;                                  //Level#
   int j = 0;                                  //Working stiff
   int max_radius = 0;                         //Max radius of a node
   int min_radius = 0;                         //Min radius of a node
   double mean_radius = 0.0;                   //Return avg. path length
   int next_node = 0;                          //Neighbor at level k+1

int n_paths = 0;                             //#paths in network
for(int i = 0; i < nNodes; i++){             //Average over all nodes
   s.init();                                 //Clear pushdown stack
   Node n0 = node[i];                        //Starting node v
   n0.level = 0; k = 0;                      //Starting level
   for(j = 0; j < nNodes; j++){              //Reset flags
      node[j].visited = false;               //Clear previous settings
      node[j].level = 0;
   }
   s.push(i);                                //Remember n0 = v
   while(s.notEmpty()){
     j = s.pull();                           //Recall n0
     node[j].visited = true;                 //Avoid circuit
     k = node[j].level;                      //Recall level
     for(int e = 0; e < nLinks; e++){        //Visit neighbors
        if(Link[ e] .head == j) next_node = Link[ e] .tail;
        else if(Link[ e] .tail == j) next_node = Link[ e] .head;
        else continue;                       //Skip over non-neighbors
        if(!node[ next_node] .visited){
          node[next_node].level = k+1;       //Assign number of hops
          node[next_node].visited = true;    //Avoid circuit
          s.push(next_node);}                //Remember next n0
     }
   }
   max_radius = 0;                           //Find largest radius
   for(int v = 0; v < nNodes; v++){
    if(node[ v] .level > max_radius) max_radius = node[ v] .level;
    if(node[ v] .level > 0) {                //Ignore all others
      n_paths++;
```

```
        mean_radius += node[v].level;          //Avg. over all non-zeros
      }
    }
    n0.radius = max_radius;                    //Maximum distance to all
   }//for
    min_radius = nNodes+1;                     //Center
    max_radius = 0;                            //Diameter
    for(int v = 0; v < nNodes; v++){
       if(node[ v] .radius > 0 &&  (node[ v] .radius <= min_radius)) {
         min_radius = node[v].radius;
         }
         if(node[ v] .radius >= max_radius) max_radius = node[ v] .radius;
      }
      if(min_radius == nNodes+1) min_radius = 0;  //Must be no path
      for(int v = 0; v < nNodes; v++) {    //Could be many centers
          if(node[ v] .radius == min_radius) node[ v] .center = true;
          if(node[ v] .radius == max_radius) node[ v] .outer = true;
      }
      reply.big = max_radius;                  //Return diameter
      reply.small = min_radius;                //Return center(s)
      if(n_paths > 0)
        reply.mean = mean_radius/n_paths;     //Return avg. path length
        else reply.mean = 0.0;                 //Return link efficiency
      reply.efficiency = (nLinks - reply.mean)/nLinks;
      return reply;
}//NW_doFindCenters
```

由于可能出现多个中心节点，而使网络中心的计算变得复杂起来，因此 Java 方法返回一组节点而非单个节点。这里给出的方法每次仅计算一个，并将所有节点半径留下来。处理过所有节点后，Java 方法将其中最小的半径节点标记成中心节点。这种集合是返回给调用函数的节点集合。

中心性定义为网络中所有路径中最大路径长度中的最小值：

$$\text{Centrality}(G) = \text{minimum}_i\{\text{maximum}_j,\{\text{length}(v_i,v_j)\}\}$$

这里我们将 length(v_i,v_j) 定义成连接节点 v_i,v_j 之间直接路径的跳数。这也是相对于 $n_0 = v_i$ 的层数 k。那么 maximum$_j$ 是这种路径中最大的，又称为节点的半径。最后，minimum$_i$ 是网络中最大路径长度中的最小值，又称为网络的中心。

为了便于说明，将层次算法应用到图 3-1 所示的例子中。观察节点 v_6 通过长度为 1 的路径连接到 4 个相邻的邻居节点上，通过长度为 2 的路径连接到 3 个相邻的邻居节点上，通过长度为 3 的路径连接到 1 个相邻的邻居节点上。对于 $n_0 = v_6$，连接到 8 个其他节点的平均路径长度为 $(4(1)+3(2)+1(3))/8 = 13/8 = 1.625$。最长的路径为 3 跳，最短的路径是 1 跳。但是 v_6 仅是网络中 9 个节点之一。我们需要计算所有节点对之间最长和最短的路径，然后在所有节点对中找到最大和最小的。这就是为什么我们要使用计算机的原因！

对于图 3-1 中的网络，$n = 9$，$m = 15$，直径 $= 4$，中心 $= 2$，平均路径长度 $= 1.777$，链路效率 $= 0.8814$。

假定每条路径长度计算复杂度为 $O(n)$，并且有多达 n^2 条路径，那么计算复杂度为 $O(n^3)$。有可能执行得更快些吗？这留给读者作为练习。

既然我们已经有一个路径长度计算器并能更好地理解链路效率，我们就可以将它们应用到本章开始所提出的问题："什么是最有效的规则网络？"一个有效的网络是稀疏的、具有小的直径并具有最短特征路径长度。

3.2 二叉树网络

线形图用 $(n-1)$ 条链路连接 n 个节点，但是它的平均路径长度为 $O(n)$。线性网络并非链路有效的，因为链路数与它的平均路径长度中的跳数增长一样快。链路更加有效的连通网络拓扑是二叉树，因为随着它的增长，它的平均路径长度增长要比链路数的增长的速度慢很多。二叉树比线性网络更加有效。

二叉树递归地定义为一个节点连接到两个也是二叉树的子树。一个节点称为根，它的度为2并且连接两个子树，后者同样连接到两个更多的子树上，依此类推。这种递归结束于一组称为叶子节点的节点上，叶子节点的度为1。所有中间节点经过三条链路连接网络，如图3-2所示。因此，二叉树包含的节点度仅可能是1，2，3。

平衡二叉树包含 k 层，并且刚好具有 $n = 2^k - 1$ 个节点，$m = (n-1)$ 条链路，对于 $k = 1, 2, \cdots$。根节点位于第1层，叶子节点位于第 k 层。每一层对应于从根到叶子节点的路径上的一跳，因此平衡二叉树的直径为 $2(k-1)$ 跳。根节点位于半径 $= (k-1)$ 的平衡二叉树的中心。

不平衡二叉树包含的节点少于 $2^k - 1$ 个。例如，在图3-2a的平衡树中，$k = 4$，$n = 16 - 1 = 15$ 节点，直径 $= 2(4-1) = 6$ 跳，$m = 15 - 1 = 14$ 条链路。但是在图3-2b的不平衡树中具有较少的节点和链路：$n = 9 < 15$，直径 $= 5$ 跳，$m = n - 1 = 8$ 条链路。在本章的其余部分仅研究平衡树。

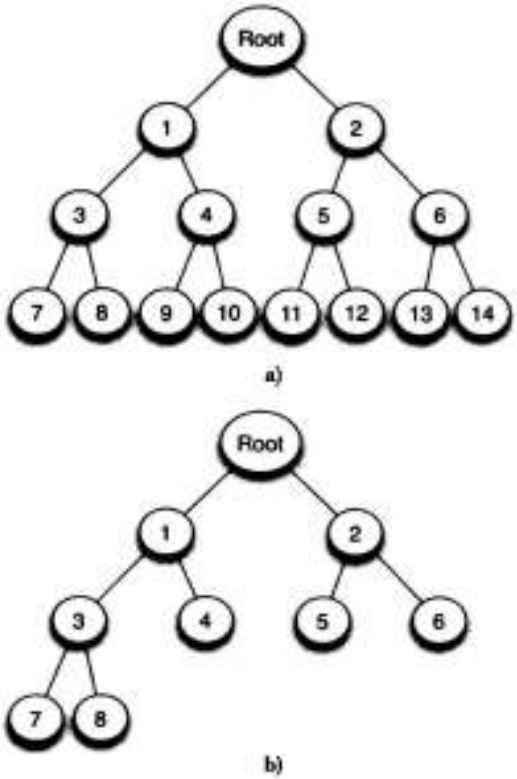


图3-2 二叉树网络：a）平衡的，$n = 2^4 - 1 = 15$ 节点；b）不平衡的，$n < 2^4 - 1 = 9$ 节点

下面Java方法创建了一个由 $n = 2^k - 1$ 个节点和 $m = n - 1$ 条链路组成的平衡二叉树。方法使用一种简单的映射函数 f：节点 i 的后继是节点 $2i+1$ 和节点 $2i+2$。Java方法简单地将所有的节点 i 连接到节点 $2i+1$ 和节点 $2i+2$ 上，但是树的最后一层节点除外。一个子树是奇数，其他的就是偶数。对于 $i = 0 \sim n/2$，奇数子树的根是节点 $2i+1$，偶数子树的根是节点 $2i+2$。

一旦根节点建立后，简单映射模式使得生成网络相当容易。给定全局参数 nInputNodes，找

到小于 nInputNodes 的最大完全平方，并生成树：

```
public void NW_doCreateTree() {
  nNodes = 1;
  int n = nInputNodes;          //Compute power of two = nNodes+1
   int log2_nNodes = 0;
   while(n > 1){
      n = n/2;
      log2_nNodes++;
      nNodes = 2*nNodes;
      }
      n = nNodes-1;
      NW_doCreateNodes(n);      //Create 2^k-1 nodes
   for(int i = 0; i < nNodes/2; i++){
      NW_doAddLink(node[i].name, node[2*i+1].name); //Left subtree
      NW_doAddLink(node[i].name, node[2*i+2].name); //Right subtree
   }
}//NW_doCreateTree
```

该代码找到小于 nInputNodes 的完全平方，然后从它的根到它的叶子节点构建二叉树。除了最后 $n/2$ 个节点外，在所有的节点上添加相同数量的左子树和右子树。这种平衡二叉树网络具有 $n/2$ 个叶子节点和 $(n/2)-1$ 个内部节点。进而，它具有 $m = n-1$ 条链路，$k = \log_2(n+1)$ 层，从 $k = 1$ 层次的根节点开始。

平衡二叉数的密度随着网络越大反而会减少。图 3-2a 中的网络密度非常低：density(平衡二叉树) = 2/15 = 0.133 。

不仅如此，密度还会随着 n 的增加而急剧减少。例如，当 $n = 1023$，$m = 1022$，密度 = 0.002 。因此，二叉树是极度稀疏的。尽管如此，它们是链路有效的，那么它们的特征路径长度短吗？答案是否定的。有可能构造出比平衡二叉树网络更有效的规则网络吗？答案是肯定的。

3.2.1 二叉树网络的熵

平衡二叉树网络是规则的，但是它的熵不为零。熵是度序列分布的函数，并且二叉树具有不平滑的度序列分布。例如，图 3-2a 中二叉树的度序列分布为

$$g' = [53\%, 7\%, 40\%]$$

这样就得出熵 I 为：

$$I(\text{平衡二叉树}, k = 4) = 1.27 \text{ 比特}$$

当然，不平滑分布的原因在于 15 个节点中有一个节点度为 2，53% 的节点度为 1，其余叶子节点度为 3。度序列总是包含三种频率：

$$p_1 = \frac{\text{度为 1 的节点数}}{n}\text{；这些是叶子节点}$$

$$p_2 = \frac{\text{度为 2 的节点数}}{n}\text{；这些是根节点}$$

$$p_3 = \frac{\text{度为 3 的节点数}}{n}\text{；这些是内部节点}$$

接近半数的节点是叶子节点，一个节点为根节点，其余节点是内部节点：

$$p_1 = \frac{n/2}{n} = \frac{n}{2n} = \frac{1}{2}\text{；叶子节点频率}$$

$$p_2 = \frac{1}{n}\text{；根节点频率}$$

$$p_3 = \frac{n-(n/2)-1}{n} = \frac{n-2}{2n}\text{；内部节点频率}$$

通过直接将上述值插入到熵的方程中得到熵 I：

$$I(\text{平衡二叉树}) = -\left[\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{n}\log_2\frac{1}{n} + \frac{n-2}{2n}\log_2\frac{n-2}{2n}\right] = \frac{1}{2} + \frac{1}{n}\log_2 n + \left(\frac{n-2}{2n}\log_2\frac{2n}{n-2}\right)$$

假定 n 值非常大，就可以进一步简化：

$$I(\text{平衡二叉树}) = 1 + \frac{\log_2 n}{n};\quad n \gg 1$$

例如，$n = 511$ 和 $m = 510$ 的平衡二叉树的熵近似地等于 $1 + \frac{\log_2(511)}{511} = 1 + \frac{9}{511} = 1.018$。

平衡二叉树网络几乎都是（但不完全是）规则网络。它在根和叶子节点是不规则的。这些“不规则性”解释了 1 比特的“随机性”。不规则性随着 n 的增加而减少，该属性在估计一般平衡二叉树网络的平均路径长度时会很有用。

3.2.2 二叉树网络的路径长度

结果证明，当它到达平均路径长度时，二叉树网络没有预期的那样有效。网络的中心是半径为 $r = k - 1$ 的根节点，叶子节点位于直径顶端节点上，直径 $D = 2(k-1)$ 跳。直径随着网络规模 n 呈对数增长，因为 $k = O(\log_2(n))$。但是平均路径长度也是按对数增长的，如图 3-3 所示。因此，二叉树的平均路径长度与它的直径成正比，如下面所示。

图 3-3 中画出了平均路径长度和 $(D-4)$ 与层 k 之间的关系图，$n = 2^k - 1$，D = 直径 = $2(k-1)$。对于高的 k 值来讲，将平均路径长度和 $(D-4)$ 合并。这样一来，平均路径长度渐近于 $(D-4)$：

$$\text{avg\_path\_length}(\text{平衡二叉树}) = (D-4);\quad k \gg 1$$

$D = 2(k-1)$，因此 $\text{avg\_path\_length} = 2k - 6 = 2\log_2(n+1) - 6$

随着平衡二叉树规模的增加，更多的叶子节点的影响远远超过接近根的极少数节点。叶子节点及其邻居占据主导地位，以至于相对较少的接近根的节点的影响减少到零。事实上，接近四分之一的节点离网络中一半以上的节点的距离为 1 跳。详细地讲，所有在层 $(k-2)$ 的节点连接到约八分之一其“上层”的节点，连接到约二分之一其“下层”的节点。显然，叶子节点和它们的直接邻居的影响主导着大的二叉树网络的平均路径长度。因此随着 n 的增加，平均路径长度渐近于 $(D-4)$ 跳。渐近于 $(D-4)$ 如图 3-3 中的点画线所示，对于大的 k 值来讲，接近于平均路径长度。

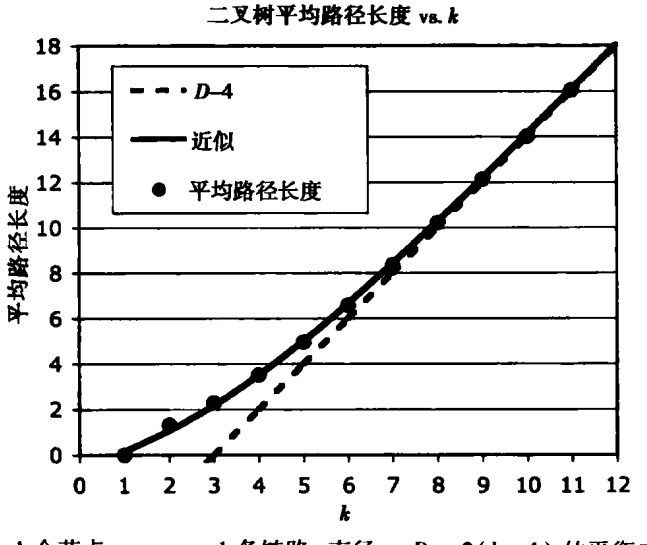


图 3-3 $n = 2^k - 1$ 个节点，$m = n - 1$ 条链路，直径 = $D = 2(k-1)$ 的平衡二叉树路径长度和 $(D-4)$ 与层 k 之间的关系图

显然，随着网络规模的提高，近似程度得到了提高。例如，$n = 32$、$m = 31$ 的平衡二叉树的实际特征路径长度为4.955。渐近逼近得出 $8 - 4 = 4$，这是对实际值的较差的估计。但是，对于 $k = 12$ 的平衡二叉树的实际路径长度为18.02，渐近方程得出 $22 - 4 = 18.00$，这是一个很好的近似。

对于较小的 k（$k < 9$）值来讲，近似不成立。相反，我们使用如图3-3所示的近似。随着 k 值的增加，近似的非线性部分呈指数减少——当（$D - 4$）占主导时达到零：

$$\text{avg\_path\_length} = (D - 4) + \frac{A}{1 + \exp(Bk)}$$

这里 $A = 10.67$，$B = 0.45$ 给出对数据的最佳拟合。$D = 2(k - 1)$，$k = \log_2(n + 1)$，代入其他数据，我们就得到

$$\text{avg\_path\_length} = 2\log_2(n + 1) - 6 + \frac{10.67}{1 + \exp(0.45\log_2(n + 1))}$$

图3-3中将这种近似显示成经过 avg_path_length 实际值的实线。对于所有 $k > 1$ 值，具有相当的精度。例如，当 $k = 1$ 时，实际路径长度为零，近似路径长度为0.15；当 $k = 4$ 时，实际值和近似值两者相同，为3.51；当 $k = 12$，实际值为18.02，近似为18.05。

3.2.3 二叉树网络的链路效率

既然我们知道平衡二叉树的平均路径长度的方程，我们可以将它插入到链路效率的方程中，并加以简化。平衡二叉树具有 $m = n - 1$ 条链路。“大的”平衡二叉树的链路效率可以简化成：

$$E(\text{平衡二叉树}) = 1 - \frac{D - 4}{m} = 1 - \frac{(2k - 1) - 4}{n - 1};\quad k > 9$$

$$E = 1 - \frac{2\log_2(n + 1) - 6}{n - 1},\ \text{因为 } k = \log_2(n + 1)$$

假定 $n \gg 1$，因此（$n + 1$）和（$n - 1$）都接近于 n，链路效率近似为

$$E(\text{平衡二叉树}) = 1 - \frac{2\log_2(n)}{n};\ k > 9$$

例如，如果 $n = 127$，那么近似地 $E = 0.890$，但是实际的链路效率为0.934。但是如果 $n = 2047$，近似公式得出 $E = 0.990$，实际的链路效率为0.992，实际和近似效率仅有0.2%的差异。对于大的 n，这种差异几乎不存在。

随着 n 无限地增长，二叉树链路效率接近100%。因此，平衡二叉树用于构造多处理器的互连或公司的组织图会显得非常有效率。在这两个应用中，每条链路需要更短的平均路径长度。平衡二叉树能以最佳效率利用链路吗？在3.3节中，我们将探索该问题并说明甚至还有比平衡二叉树网络更有效的网络。

3.3 超环形网络

二叉树平均路径长度在很大程度上是由从根节点到它的叶子节点的距离的对数增长所决定的。为了从一个叶子节点到达另外一个叶子节点，信息必须沿着树到达它的根，然后再向下到达目的地。两个叶子节点越远，路径就越有可能通过树根。

或许通过在二叉树同一层节点之间插入横向链路，可以减少平均路径长度。这些捷径将不需要遍历所有的路径到根节点，而是简单地从树的一侧到达另外一侧。但是，增加链路会降低链路效率。因此，在不增加更多的链路时如何缩短路径？一种设计是在网格状的拓扑中将一半链路分给水平连接，另外一半分配给垂直连接。换句话来讲，假设映射函数连接直接前驱和后继节点，另外一个映射连接节点 v 到另外一个距离为 $+\sqrt{n}$ 和 $-\sqrt{n}$ 跳的节点，如图3-4所

示。这种网格似的到节点的链路映射使用少量的链路，同时降低了节点间的距离。这导致了曼哈顿似的网格。

图3-4所示的网络是一个 $\sqrt{n} \times \sqrt{n}$ 的网状，并附加了一个条件：边沿节点“围绕”到相对的边沿节点，因此每一行和列构成一个环形。这就保证短路径，原因就像前一章中所见，环形图的平均路径长度为 $O(n/4)$，而线形图的平均路径长度为 $O(n/3)$。当像这样绕起来时，环的集合就构成一个超环形网络，这种网络故此得名。

典型的超环形网络具有 $n = k^2$ 个节点，以 $k \times k$ 网格布置。每个节点的度为4，因此有 $m = 2n = 2k^2$ 条链路。图3-4显示了一个 4×4 网络，具有 $n = 16$ 个节点和 $m = 32$ 条链路。总的来讲，超环形网络的规模等于整数的平方（$n = 4,9,16,25,36,\cdots$），链路数是整数平方的两倍。

图3-4的超环形网络是完全非随机的，因为它的熵 I(toroid) 为零。这是由于实际上所有节点具有相同的度：degree(toroid) = 4。它也缺乏聚类，cc(toroid) = 0，因为节点的邻居之间不相互连接。所有的节点在网络的中心；对于网络中的所有节点 v 来讲，diameter(toroid) = 4，radius(v) = 4。

由于其规则性，构建一个超环形网络非常容易。难度仅来自于垂直和水平围绕链路。垂直围绕由 n 的模函数来处理，水平围绕由 $\sqrt{n}$ 的模函数处理。从另一个角度来看，要注意每一行都是带有围绕链路的行，围绕链路的第一行在节点零和节点号 $\sqrt{n}$ 之间，第2行在 $\sqrt{n}+1$ 和 $2\sqrt{n}$ 之间，依此类推。相似地，垂直环连接被 $\sqrt{n}$ 中间节点分隔开的节点，并从 $\sqrt{n}$ 行到1行围绕。

一般来讲，下列映射函数控制着超环形的拓扑：

$$f: v_i \rightarrow v_{(i+1) \bmod \sqrt{n}}; \quad v_i \rightarrow v_{(i+\sqrt{n}) \bmod n}$$

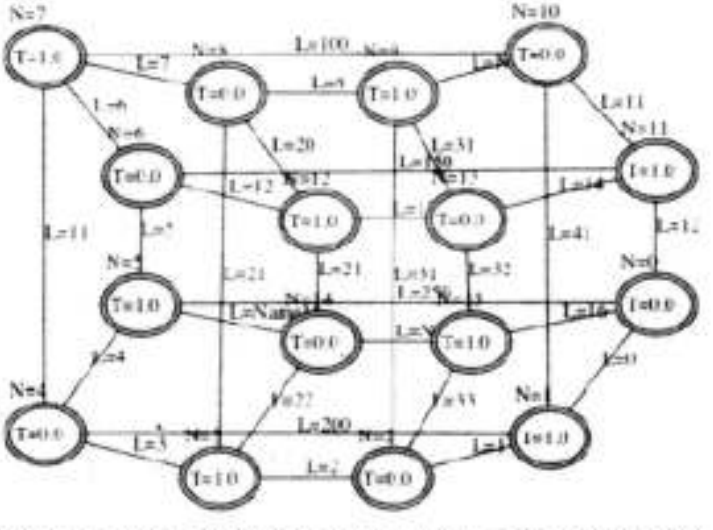


图3-4 从程序 Network.jar 的屏幕中显示了一个 $n = 16$，$m = 32$ 的超环形网络，平均路径长度为2.13跳

在找到小于 nInputNodes 的最大完全平方后，计算 sqrt(n) 后，用以下 Java 方法实现这种映射函数：

```
public void NW_doCreateToroid() {
  int sqrt_nNodes = (int)Math.sqrt(nInputNodes); //length of a side
  int n = sqrt_nNodes * sqrt_nNodes;             //Perfect square
  NW_doCreateNodes(n);                           //Create k^2 nodes
  for(int row = 0; row < nNodes; row += sqrt_nNodes){
   for(int col = 0; col < sqrt_nNodes; col++){   //Links
    int i = row + col;
    int j = col + 1;
    if(j >= sqrt_nNodes) j -= sqrt_nNodes;       //Wrap horizontal
    j += row;                                    //Add next node link
    NW_doAddLink(node[i].name, node[j].name);    //Add horizontal link
    j = i + sqrt_nNodes;
    if(j  >= nNodes) j -= nNodes;                //Wrap vertical
    NW_doAddLink(node[i].name, node[j].name);    //Add vertical link
    }
  }
}//NW_doCreateToroid
```

这种方法是从网格的左上角的节点零开始，并将每一行从左到右的节点连接到它的直接后继，及它的后继 + sqrt(n) 之前。在每一行的结束，该方法通过将每行最后一个节点连接到第一个节点形成了一个环。当到列的最后时，该方法通过将各列的最后一个节点连接到第一个节点构成另外一个（垂直）环。这些环使得超环形网具有较短的平均路径长度。

3.3.1 超环形网络的平均路径长度

下面，我们演示超环形网络的平均路径长度会随着网络规模 n 按照 $O(\sqrt{n})$ 增加。这比二叉树网络的路径长度要小，从而导致更好的链路效率。例如，4 × 4 超环形网络的平均路径长度为 2.133 跳，而 n = 15 个节点的二叉树具有 3.5 跳。任意 $k \times k$（$n = k^2$）超环形网络的平均路径长度是多少？为了回答该问题，考虑表 3-2。

表 3-2 超环形网络的路径矩阵分析结果

规模，n	超环形	行总和	因数分解行总和	平均路径长度
4	2 × 2	4	2 * 2	4/3 = 1.33
9	3 × 3	12	3 * 4	12/8 = 1.50
16	4 × 4	32	4 * 8	32/15 = 2.13
25	5 × 5	60	5 * 12	60/24 = 2.50
36	6 × 6	108	6 * 18	108/35 = 3.09
49	7 × 7	168	7 * 24	168/48 = 3.50

表 3-2 是通过枚举规模为 n = 4,9,16,… 的超环形网络的路径长度从路径矩阵构造的。这些是完全平方数，因此 $\sqrt{n}$ = 2,3,4,… 。平均路径长度等于路径矩阵的所有 $n(n-1)$ 个元素的平均，但是路径矩阵的所有行的和都等于同一数，因此很容易计算所有行的平均值。

表 3-2 的第 3 列包含每个超环形规模的行总和，第 4 列也是行总数，但是以因数分解形式表示。例如，一个 4 × 4 超环形路径矩阵的行之和均等于 32，它的因数分解为 4（8）。注意 4 = $\sqrt{16}$ 。这样一来，第 4 列可被分解成 $\sqrt{ns}$ ，这里当 n 为奇数时，s 为 $(n-1)/2$ ；当 n 为偶数时，s 为 $n/2$ 。对于 n = 16 ，$s = \frac{16}{2} = 8$ ，因为 n 为偶数。对于 n = 25 ，s = (25 − 1)/2 = 12 ，因为 n 为奇数。这样一来，行总和（表 3-2 中第 4 列）的因数分解一般形式为：

$$\text{当 } n \text{ 为奇数时，行总和} = \sqrt{n}\,\frac{n-1}{2}\text{；当 } n \text{ 为偶数时，行总和} = \sqrt{n}\,\frac{n}{2}$$

接下来，观察表 3-2 的最后一列，包含以分数形式 a/b 表示的平均路径长度，其中分子 a = 行总和，分母 b = $(n-1)$ 。例如，当 n = 16 时，分子 a = 32 ，分母 b = (16 − 1) = 15 。因此

4 × 4 超环形网络的平均路径长度等于 32/15 = 2.13 。通过推导，平均路径长度的公式为：

$$\text{avg\_path\_length(toroid)} = \begin{cases} \frac{\sqrt{n}}{2}, & n \text{ 为奇数} \\ \sqrt{n}\frac{n}{2(n-1)}, & n \text{ 为偶数} \end{cases}$$

读者可以验证该公式适用于任何规模的超环形网络。奇数规模的超环形网络稍稍紧凑一些，但是偶数和奇数网络的平均路径长度都是按照 $O(\sqrt{n})$ 增长的。这种近似随着 n 的增加快速得到改进。例如，程序 Network. jar 和该公式对于 12 × 12 超环形网络都给出 6.042 的平均路径长度。在本章稍后我们将演示 $n > 16$ 的近似值非常接近精确值。为了实用的目的，可以简单地说超环形网络的平均路径长度为 $(\sqrt{n})/2$ ，$n > 16$ 。

3.3.2 超环形网络的链路效率

如图 3-4 中所示的一个超环形设计，部分地满足一个有效网络是稀疏的并有效地利用链路以便降低平均路径长度的要求。超环形网络的效率仍受相对较大的链路数的制约。链路数 $m = 2n$ 按照 k^2 增加，因为 $n = k^2$ 。这在某些应用中这可能是一个缺点。例如在计算机体系结构中，$m = 2k^2$ 条链路布局在一维平面上，如硅晶片，就限制了超环形网络体系结构采用相对少量处理器节点。在本节中，我们已经说明了超环形的链路效率要比二叉树的好，但是这还不够，因为超环形网络使用 $m = 2n$ 条链路实现相对小的平均路径长度 2 [⊖]。

将平均路径长度的表达式带入到链路效率的方程中，并简化得出超环形网络的链路效率：

$$E(\text{toroid}) = \frac{(m-\sqrt{n})/2}{m} = 1 - \frac{\sqrt{n}/2}{2n} = 1 - \frac{\sqrt{n}}{4n} = 1 - \frac{1}{4\sqrt{n}}$$

将 $n = 16$ 的超环形网络与 $n = 15$ 的二叉树网络的链路效率进行比较，得出 $E(\text{toroid}) = 1 - \frac{1}{16} = \frac{15}{16} = 93.8\%$ ，$E(\text{binary tree}) = 1 - (8-6)/14 = \frac{12}{14} = 85.7\%$ 。对于较小的 n 来讲，相差 9%。随着 n 的增加，超环形网络仅比二叉树网络效率稍高，因为随着 n 的增加，百分比之差下降了。例如，当 $n = 100$ 时，超环形链路效率是 97.5%，二叉树链路效率近似于 92.6%，只有 5% 的差距。但是，超环形需要二叉树两倍的链路。

3.4 超立方网络

*超立方网络*是以节点间相距只有 1 个海明距离的方式连接节点。两个二进制数 x 和 y 之间的海明距离 $h(x,y)$ 定义成 x 中的比特与 y 中对应比特不同的个数。因此，仅对于 $h(x,y) = 1$ 的节点，超立方通过将节点 v_x 和 v_y 连接起来而形成。

表 3-3 二进制数之间的海明距离[①]

海明距离	000	001	010	011	100	101	110	111
000	0	1	1	2	1	2	2	3
001	1	0	2	1	2	1	3	2
010	1	2	0	1	2	3	1	2
011	2	1	1	0	3	2	2	1
100	1	2	2	3	0	1	1	2

注：①列，0，…，7；行，0，…，4 。

表 3-3 列出了整数 0 ~ 7（表 3-3 中的列）的二进制表示和整数 0 ~ 4（表 3-3 中的行）的二

⊖ 布局不是平面的，也就是说不能够在二维平面上不重叠链路地画出。

进制表示之间的海明距离。例如，整数2的二进制表示为010，4的二进制表示为100，它们的比特位中有两个二进制位数不同，因此海明距离为2。

海明距离可以通过将由XOR（异或）操作符产生的比特加起来计算。当输入比特不同时XOR产生“1”比特，当输入相同时XOR产生“0”比特：

$$XOR(010,100) = 110$$

$$SUM(110) = 2$$

为了构建一个超立方，简单地将相邻节点对v_x和v_y连接起来，如果$h(x,y) = 1$，就在v_x和v_y之间画一条链路。假设$n = 4$，那么节点索引范围为0~3，以二进制表示为：00，01，10，和11。我们使用相差一比特位的二进制索引将所有节点对连接起来。在这种情况下，$h(00,01) = 1$，$h(00,10) = 1$，$h(01,10) = 2$，$h(01,11) = 1$，$h(10,11) = 1$，因此我们将等于1的所有节点对连接起来如下：

节点	二进制索引	链路
0	00	00~01 00~10
1	01	01~00 01~11
2	10	10~00 10~11
3	11	11~01 11~10

图3-5b显示了通过这种映射获得的结果网络。在00~11内没有链路，因为在节点0和节点3之间的海明距离不是1：

$$XOR(00,11) = 11,\ SUM(11) = 2$$

超立方的维数等于每一个节点的度，$D(H) = \log_2(n)$，这里$n = 2^D$。二维超立方的所有节点具有度2，三维超立方的所有节点具有度3，以此类推。超立方的直径也等于节点的度。因此，维数、直径和节点的度都是相同的。

像嵌套的俄罗斯套娃一样，超立方网络中包含超立方网络！维数为D的超立方包含两个维数为$D/2$的超立方，如图3-5所示。从$D = 1$的超立方开始，图3-5a的一维杠铃很容易从两个节点构造——一个序号为0，另外一个序号为1。$D = 2$的超立方如图3-5b所示，通过用两条新的链路将两个杠铃超立方结合起来构造。更高维数超立方的二进制节点索引是通过在一个杠铃的节点索引添加二进制0，在另外一个杠铃的节点索引添加二进制1获得的。在该例子中，节点索引00、01是通过在索引号为0和1的节点添加前缀“0”获得的，节点10、11通过从第二个杠铃的节点索引号为0和1的节点添加前缀“1”获得的。

一般来讲，更高维数的超立方的建立方法是通过复制两个（$D-1$）维超立方，在一个副本的节点索引前加前缀二进制0，另一个副本的节点索引前加前缀二进制1，并在索引仅相差一个海明距离的节点添加链路。图3-5演示了这项技术。

超立方的每一个节点连接到$D = \log_2(n)$的其他节点上，每条连接只有一个海明距离。例如，当$n = 4$时，那么$\log_2(4) = 2$，因此节点00连接到节点01和10。这两个邻接节点距离00节点仅有一步海明距离。邻接节点索引是通过翻转00的每一比特位计算的，每次一位，这样节点索引为01和10。相似地，对于$n = 8$，$D = \log_2(8) = 3$，在000和100、010和001之间插入了三条链路。通过翻转000的第1、2、3比特位，一次一位，相邻节点索引为100、010和001。

超立方生成过程简单，对于超立方中的每一个节点 v_i，在 v_i 和所有其他节点之间插入链路，节点索引是从 i 的每一比特翻转获取的，每次一位，从左到右直到所有的 $\log_2(n)$ 比特翻转完为止。

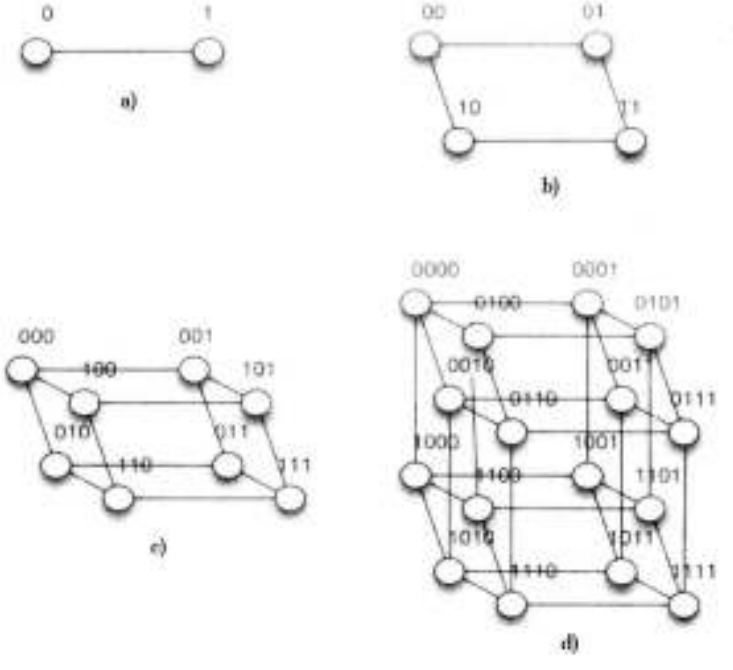


图 3-5 超立方网络：a) 杠铃（$n=2$）；b) 环形（$n=4$）；c) 3D（$n=8$）；d) 4D（$n=16$）配置

构建 $n=2^p$ 个节点的超立方网络的 Java 代码显示如下。类 BitOps 包含三种特殊意义的方法：toString() 将一个整数转换成一个二进制字符串，toInteger() 将一个二进制字符串转换成一个十进制整数，flipBit() 将二进制字符串的第 j 个比特从 0 变成 1，从 1 变成 0。

```
public void NW_doCreateHypercube() {
    BitOps h = new BitOps();
    nNodes = 1;
    int n = nInputNodes;                    //nNodes = 2^k
    int log2_nNodes = 0;                    //Number of bits in index
    while(n > 1){
      n = n/2;
      log2_nNodes++;                            //degree, D
      nNodes = 2*nNodes;
    }
    n = nNodes;                             //n = 2^k
    NW_doCreateNodes(n);                    //Create 2^k nodes
    for(int i = 0; i < nNodes; i++){     //For each node...
       String iString = h.toString(i, log2_nNodes);   //Node index as a bit string
       for(int j = 0; j < log2_nNodes; j++){
          NW_doAddLink(          //...add log2(n) links
          node[i].name,
          node[h.toInteger(h.flipBit(iString, j))].name);
       }
    }
}
```

方法 NW_doAddLink() 拒绝重复链路，因此我们不必担心由于将节点 00 连接到 10、将 10 连接到 00 所引起的重复。但是，如果 NW_doAddLink() 不能避免重复，需要对方法 NW_doCreateHypercube() 做出何种修改才能避免重复呢？这一问题留给读者作为练习。

3.4.1 超立方网络的平均路径长度

回顾一下，任何网络的链路数目等于节点度总和的一半：$2m = \sum_1^n d_i$。超立方网络的每个节点具有同样的度，$D = \log_2(n)$。这样一来，$m = n(\log_2(n))/2$，这里 $n > 1$。在图3-5d的4D超立方中有 $m = 16\left(\frac{4}{2}\right) = 32$ 条链路。在5D超立方中有 $m = 32\left(\frac{4}{2}\right) = 64$ 条链路，因此链路数随着超立方规模而扩展。

超立方中的每一个节点都可以以 $1,2,\cdots,D = \log_2(n)$ 跳从任何一个其他节点到达。对于所有节点都是如此，因此所有节点都是中心节点，所有节点的半径等于网络的直径 D。因此，平均路径长度为：

$$\text{avg\_path\_length(hypercube)} = \frac{n\log_2(n)}{2(n-1)}$$

例如，图3-5中的四个超立方网络对于 $n = 2,4,8,16$ 的平均路径长度分别为1.0、1.33、1.714和2.133。假定 $n \gg 1$，我们可以将公式简化为：

$$\text{avg\_path\_length(hypercube)} = \frac{\log_2(n)}{2};\quad n \gg 1$$

该方程是利用所有超立方的对称性导出的。每一个节点的平均路径长度与所有其他的节点相同，因此我们仅计算节点000的平均路径长度：

1. 所有节点的平均路径长度都是相同的。因此，从节点000到所有其他 $(n-1)$ 个节点的平均路径长度也是整个网络的平均路径长度。

2. 对于 $n = 8$，计算节点000的平均路径长度，然后归纳到 $n > 1$ 的情况。特别是，注意从000到所有其他 $(n-1)$ 个节点有 $(n-1)$ 条路径，路径长度数 D 等于二项式系数的总和，$B = \sum_0^{D-1} C_i^{D-1}$，这里 $C_i^{D-1} = \frac{(D-1)!}{i!(D-i-1)!}$，换句话来讲，$C_i^{D-1}$ 是通过枚举 $(D-1)$ 获得的组合系数，每次取出一个 i。

3. 二项式系数和 $B = 2^{D-1} = n/2$，因为 $n = 2^D$。

4. 最后，假定 $n \gg 1$，简化平均路径长度的一般表达式。结果为

$$\text{avg\_path\_length(hypercube)} = (\log_2(n))/2; n \gg 1$$

考虑图3-5c中显示的 $n = 8$ 和 $D = 3$ 时的情况。我们计算节点000的平均路径长度，通过将路径长度数1加上长度数2及加上长度数 $D = 3$，并被 $(n-1) = 7$ 除得到：

$$\text{avg\_path\_length} = \frac{3(1)+3(2)+1(3)}{7} = \frac{3(1+2+1)}{7} = \frac{12}{7} = 1.714$$

所有节点具有同一路径长度，因此该表达式也是整个3D立方网络的平均路径长度。通过将3替换成 D 以及7替换成 $(n-1)$ 可生成以下表达式，并且注意项 $(1+2+1)$ 刚好等于组合系数之和 $(C_0^2 + C_1^2 + C_2^2)$：

$$\text{avg\_path\_length} = D\frac{C_0^2 + C_1^2 + C_2^2}{n-1} = D\frac{\sum_0^{D-1} C_j^{D-1}}{n-1} = D\frac{2^{D-1}}{n-1} = n\frac{\log_2(n)}{2(n-1)} = \frac{\log_2(n)}{2};\quad n \gg 1$$

例如，图3-5d的4D超立方的平均路径长度为2.133，因为 $n = 16$：

$$\text{avg\_path\_length} = 16\frac{\log_2(16)}{2(16-1)} = 8\frac{4}{16} = \frac{32}{15} = 2.133$$

平均路径长度是直径的一半，它也是维数的一半。这样一来，平均路径长度随着超立方维数而扩展，它是 $O(D)$ 和 $O(\log_2(n))$：

$$\text{avg\_path\_length(hypercube)} = \frac{D}{2} = \frac{\log_2(n)}{2}; \quad n \gg 1$$

这种结果可以通过运行程序 Network. jar 实验证实。

3.4.2 超立方网络的链路效率

超立方网络是非常有效的，因为随着 n 的增加，它们会呈对数地稀疏。链路数 $m = n(\log_2(n))/2$ 是以 $O(n\log_2(n))$ 增长的，而平均路径长度是以 $O(n\log_2(n))$ 增长的。因此，很容易计算链路效率如下：

$$E(\text{hypercube}) = 1 - \frac{\text{avg\_path\_length(hypercube)}}{m} = 1 - \left[n\frac{\log_2(n)}{2(n-1)m}\right]$$

$$= 1 - \left[n\frac{\log_2(n)}{2(n-1)n\dfrac{\log_2(n)}{2}}\right] = 1 - \frac{1}{n-1}$$

例如，图 3-5c 中的 3D 超立方的链路效率为 $1 - \frac{1}{7} = 0.857$，图 3-5d 中的 4D 超立方的链路效率为 $1 - \frac{1}{15} = 0.933$。随着网络规模的增大，它就变得更加有效。因此超立方网络是可扩展的。

超立方网络是本章研究的最有效的规则网络。从链路效率来讲：$E(\text{hypercube}) > E(\text{toroid}) > E(\text{binary tree})$。这对于所有规模的网络都有效吗？图 3-6 显示了这三种规则网络的平均路径长度和链路效率与规模 n。

如此一来，$n > 16$ 时超立方要比二叉树更加有效，而且要比超环形网络更加有效。但是当 $n > 16$ 时，超立方要比环形网络使用更多的链路。这可以解释为什么以超立方方式互连的网络用于“小的”多处理器系统中，而超环形网络用在“大的”系统中。

注意 $n > 6$ 的超立方和超环形网络链路效率的微小不同。似乎两者是可比的，但是超立方具有更小的平均路径长度并且更有效率，尽管超环形使用更少的链路。

有利于“小”超立方系统的另外一方面在于，较小 n 值的超立方系统效率会快速提高，如图 3-6 所示。这可能解释为什么超立方拓扑经常用于 8、16 和 32 块处理器的多处理器互连网络中。

规则性在许多网络应用中是一个优点。规则网络具有零或者非常小的熵和小的聚类。令人惊讶的是，它们也可以非常有效，如图 3-6 所示。稀疏网络如超立方可能会同样有效，甚至要比接下来将要学习的随机网络、小世界网络和无标度网络更加有效。

总之，规则网络在许多学科如计算机设计、有限元分析、理解材料的晶体结构及建筑物建模方面具有重要意义。但是在网络科学中，规则网络更多的是作为构造更复杂网络的起点。例如在第 5 章中，我们将阐述如何从 k - 规则和超环形网络开始构造小世界网络。

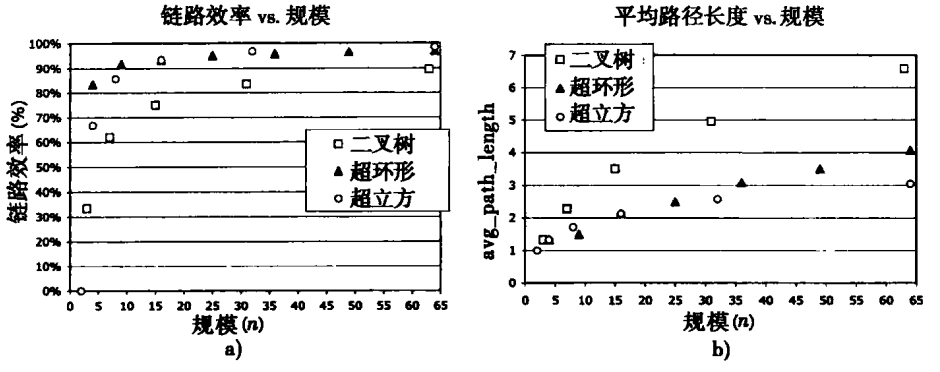


图 3-6 二叉树、超环形和超立方网络：a）实际链路效率的比较；b）平均路径长度

练习

3.1 导出具有 $n = 2^k - 1$ 个节点的平衡二叉树的密度通用公式。

3.2 平衡二叉树网络的熵不为零并随着网络规模的增大而降低。随着 n 无限制地增大，平衡二叉树网络的熵的极限是多少？

3.3 $n = 511$ 个节点的平衡二叉树网络的平均路径长度是多少？$n = 1023$ 时如何？实际值与本章中导出的近似公式匹配程度如何？

3.4 $n = 1023$ 个节点的平衡二叉树网络的近似链路效率是多少？这种估计的准确性如何？

3.5 $n = 1024$ 个节点的超环形网络的平均路径长度是多少？这种估计的准确性如何？

3.6 导出 $n = k^2$ 个节点的超环形网络的密度计算的通用方程。

3.7 确定并解释超环形和超立方网络的熵和聚类系数的来源或起因。

3.8 $n = 512$ 个节点的超立方网络的平均路径长度是多少？实际值与本章中导出的近似公式的相关性如何？

3.9 $n = 16$ 的超立方网络的近似链路效率是多少？这种近似的精确度如何？

3.10 n 个节点的超立方网络的密度是多少？

3.11 修改 NW_doCreateHypercube() 的 Java 代码以避免重复链路。解释你的修改是如何避免节点 000 和节点 100 之间的重复链路的？

3.12 本章给出的计算平均路径长度的算法的时间复杂度为 $O(n^3)$。应该如何改进以便降低计算的复杂度，并解释为什么你的修改会有所提高。

3.13 网状网络是一种没有围绕链路的超环形网络。因此，网状网络具有 $m = 2\sqrt{n} - 1$ 而不是 $2n$ 条链路。规模为 n 的网状网络的平均路径长度的通用公式是什么？它的链路效率如何？同等大小的超环形网络和网状网络哪一个更有效？

3.14 规模为 n 的 2 - 规则网络的近似平均路径长度方程是什么？

3.15 对于 $4 \leqslant n \leqslant 9$，二叉数、超环形或超立方网络中哪一种网络具有最短的平均路径长度？

随机网络

随机网络是最早被研究的网络之一，可以追溯到20世纪50年代的有关图论文献。但是，这并不能代表自然界本身就是随机的。实际上，我们发现刚好与此相反，大多数物理和生物系统不是随机分布的，而是包含结构的。那么为什么还要研究随机网络？因为在与更加结构化的网络对比时，可以将随机网络类作为基线。在问某种网络如何区别于随机网络之前，我们就必须理解随机网络的各种属性。然后，我们就可以将这些属性与小世界网络和无标度网络的同等属性进行比较。当与随机网络相比时，大多数实际网络会有很大的不同。表2-3演示了随机网络和非随机网络之间的显著区别。

随机网络在结构化频谱中刚好与结构化网络相对。对于大的 n，随机网络用泊松度序列分布来描述，而对于小的 n 则用二项式分布来描述。例如“随机拓扑”导致高熵，这是随机网络的一个标志性属性。稀疏随机网络也显示小世界效应——它们的直径随着少量随机链路的添加而快速收缩。但是随机网络的其他属性则相当不显著，例如聚类系数和介数/紧度属性与某些结构化网络相比就比较适度。这就支持采用随机网络作为基线的思想。

我们给出两个主要生成过程，并建议对 Erdos-Renyi（ER）算法稍做修改以便生成随机网络（Erdos, 1960, 1976a, 1976b）。一个生成过程（Gilbert）从一个完全网络开始，然后删除随机选择的链路，直到获得需要的链路密度为止。生成随机网络的另外一个过程（ER）是通过在随机选择的节点对之间插入链路直到到达需要数量的链路为止。Gilbert 和 ER 过程两者以非零概率产生不连通的网络，因此第三个生成过程（锚定的 ER）提供一个以牺牲某些随机性为代价来保证的连通图。

为了保持本书所应用的方法与网络科学一致，我们生成实用的证据以支持度序列分布、平均路径长度、直径和聚类系数属性的理论结果。我们演示了随着链路增加，熵从零增加到最大值，然后再回到零。这种非直觉观察可以很容易地解释：随着随机网络的密度接近（结构化）完全网络，随机网络变得不随机了。

本章再次介绍介数/紧度属性，并提供计算紧度的 Java 代码。回顾紧度是一种对中间节点（中介过渡）的有用性的测量。如果很多最短路径通过邻近的节点与其他节点相链接，那么它就被认为是接近其他节点。我们提供计算最短路径的方法，并用对应于穿越每个节点的路径数值标记节点。

本章介绍并描述以下内容：

1. 两个具有历史意义的网络：由 Gilbert 建议的第一个随机网络，以及由 Erdos 和 Renyi（ER）建议的更著名的算法。我们将演示这些网络几乎具有同样的属性，即使它们是由两种不同的微规则集合产生的。

2. 生成过程（和对应的 Java 方法）是为构造 Gilbert、ER 和锚定随机网络而给定的。锚定随机网络稍低于完美的随机网络，但是减少了孤立节点的出现。

3. 随机网络的熵是密度的函数，这里密度为 $2m/(n(n-1))$。随机网络仅对于中等的密度值是随机的——对于低或高密度它们就不是随机的。具有50%密度的随机网络是“最佳随机”，

因为该密度最大化了熵。我们演示本章所研究的小网络的熵是根据如下公式变化的：

$$I(\text{random}) = 4 - 2[\exp(-13(\text{density})) + \exp(-13(1-\text{density}))];0 < \text{density} < 1$$

4. 随机网络是高度链路有效的，因为少量的链路增加就会对平均路径长度的下降（小世界效应）有很大影响。从理论上讲，随机网络的直径为 $O(\log(n))/\log(\lambda)$，这里 λ 为网络中节点的平均度。随机网络的平均路径长度根据其密度而变化：

$$\text{avg\_path\_length} = \frac{O(\log(n))}{\log(n(\text{density}))} = \frac{A\log(n)}{\log(Cn(\text{density}))}$$

5. 对于随机网络来讲，聚类系数近似于 CC = 密度 = λ/n，即聚类直接与添加到网络上的（随机）链路数成正比。

6. 随机网络的直径和半径随着网络密度的增加而快速降低，它们服从与平均路径长度相同的关系，只不过统一模型

$$\frac{A\log(n)}{\log(Cn(\text{density}) + D)}$$

中的参数 A、C 和 D 值不同。随着密度的增加，随机网络快速变得“更小”——它们显示出小世界效应。

7. 平均紧度的节点的度随着密度的增加而增加，达到最大值后，沿着最短路径随着密度和平均节点数的增加而减少：

$$\text{closeness(random)} = O((1-\text{density})\lambda'),\lambda = \text{平均度},r = O\left(\frac{\log(n)}{\log(\lambda)}\right)$$

此外，通过最近节点的路径数会根据“路径数(intermediary) = O(avg_path_length)”而变化，显示出紧度与小世界效应之间的关联。

8. 最后，本章探索了随机网络的“六度分隔”现象。我们将演示弱联系的 Granovetter 概念（Granovetter, 1973）作为对应于社会网络的概念，并演示随机网络中的社会联系与网络直径成正比。本章使用程序 Network. jar 生成并分析这里描述的每一个网络。实现本章中给出的生成过程的 Java 方法是 Network. jar 中找到的代码的简化版本。

4.1 随机网络的生成

生成无重复链路、无循环、无孤立节点或多组件的随机网络并非那么简单。一方面我们想要使生成的网络是随机的（高熵），另外一方面我们想要避免可能由于生成过程“意外”出现的孤立节点、重复链路和循环。这需要在设计中以及实现生成这种网络的微规则中加以注意。

首先，我们检验 E. N. Gilbert 在 1959 年建议的随机网络生成规则（Gilbert, 1959）。然后再研究 Erdos 和 Renyi 建议的随机网络生成微规则（Erdos, 1960）。ER 微规则产生著名的 ER 网络。Gilbert 和 ER 过程会产生可能包含孤立节点和多个组件的网络。因此，我们提供第三种算法，至少锚定一条链路到每个节点上。所建议的锚定生成过程保证所有的节点至少连接到一个其他节点上，但是却牺牲了随机性。

总的来讲，所有生成过程试图随机化网络的度序列分布的网络。“完全随机”网络就是一种服从二项式度序列分布的网络。但是当然，随机地生成网络是这种思想的唯一近似。我们演示为什么随机网络在小 n 值时其度序列为二项式分布，对于大 n 值来讲则为近似泊松分布。

4.1.1 Gilbert 随机网络

Gilbert 随机网络生成过程的基本思想是以概率 p 从 n 个节点的完全图中选择链路，以便于结果网络平均以 $m = p[n((n-1)/2)]$ 条链路结束。换句话来讲，Gilbert 网络的密度为 p，因为完全网络具有 $n((n-1)/2)$ 条链路。

另外一种方式是将 Gilbert 网络看做是从具有 n 个节点和 m 条链路的 $C\binom{n}{2}$ 中的可能之一，这里 $C\binom{n}{2}$ 为组合函数，定义如下：

$$\frac{n!}{2!(n-2)!} = n\frac{n-1}{2}$$

例如，如果 $n = 100$，那么

$$C\binom{100}{2} = \frac{100!}{2(98!)} = \frac{100(99)}{2} = 4950$$

如此一来，Gilbert 网络是从 4950 种可能网络中随机选择的一种。

给定 n 和概率 p，应用如下微规则便产生一个 Gilbert 网络：

1. 初始化：生成 n 个节点并从 0 到 $(n-1)$ 进行编号。
2. 设置 m：使 $m = n((n-1)/2)$ = 完全图中的节点数量 。
3. 对于 $i = 0,1,\cdots,(m-1)$ 重复进行以下工作：
   a. 假定 Math. random() $< p$，将链路 i 连接到一对节点上；否则，忽略。
   b. 计算链路连接数并计算密度：

$$\text{density} = \frac{\text{连接的链路数}}{m}$$

这种生成过程使用区间［0，1）内的一个随机数，以便确定是否将一条链路插入到网络中。如果实验 Math. random() $< p$ 失败，那么就没有链路插入。过程测试 $n((n-1)/2)$ 条链路中的每一条，并随机地将 $p[n((n-1)/2)]$ 条链路插入到节点对之间。但是具体是哪一对节点呢？任何节点对的选择方法都是可以接受的，因此下面的 Java 方法采用一种容易的方法——简单地列举所有 $n((n-1)/2)$ 个节点对，然后随机地选择链接的节点对。这种方法避免了错误地插入重复链路的风险，因为对每个节点对仅访问一次。它也避免了循环，因为跳过了对角线——仅当任何节点对相同时才会发生。

注意插入链路数随着 p 的增加而增加，因为 p 实际上是对密度的测量。Gilbert 的方法产生一个具有某一密度的随机网络。同 Erdos-Renyi 随机网络相对比，Gilbert 网络使用尽可能多的链路保证密度近似等于 p。密度是直接可调整的，并确定链路数目作为副产品。平均来讲，这些是对等的，但是根据特殊随机网络，具体链路数会因网络不同而异。

创建 Gilbert 网络的 Java 方法（下面给出）调用 NW_doCreateNodes()，它初始创建 nNodes 节点。然后，NW_doCreateGilbert() 从 $n((n-1)/2)$ 可能的链路中随机地选择大约 $p[n((n-1)/2)]$ 条链路，并将它们插入到网络中。全局变量 nConnectionProbability 为 p，由它确定网络的密度。

在任何通过这种方法创建的网络中，由于从随机数产生器取样而导入的不确定性，实际的链路数会不同。由于每次 Gilbert 网络生成时最后的链路数会有所不同，方法将返回插入的链路数。这个数存储在变量 count 中，根据密度公式，这与密度相关：

```
private void NW_doCreateGilbert() {
    NW_doCreateNodes(nInputNodes);          //Create nodes, only
    int count = 0;                          //How many links?
    for(int i = 0; i  < nNodes; i++){
        for(int j = 0; j  < i; j++){
```

```
            if(i == j) continue;              //Skip self
            if(100*Math.random() < nConnectionProbability)
             if(NW_doAddLink(node[ i] .name, node[ j] .name, Long.toString(nLinks)))
                        count++;
        }
    }
}//NW_doCreateGilbert
```

方法 NW_doAddLink(node[i].name, node[j].name, s)在节点 i 和 j 之间创建一条新的链路，带有文本标记 s，并且如果没有链路存在就返回 true，否则返回 false。但是当然，这种过程通过仅选择节点对一次而避免了重复[⊖]。这种方法的 Java 代码如下：

```
public boolean NW_doAddLink(String from, String to, String name) {
    if(from == to) return false;      //No self loops
    Link e = new Link();              //Allocate new link
    e.tail = NW_doFindNode(from);     //Node pair...
    e.head = NW_doFindNode(to);
    e.value = nDefaultValue;          //Default link value
    e.name = name;                    //Link name
    boolean exists = isConnected(e.tail, e.head);
     //Link could point either direction
    if(!exists) {                     //OK to insert new link
        Link[nLinks] = e;             //Insert in list of links
        Link[nLinks].c = Color.black;//Paint Black
        nLinks++;
        node[e.tail].degree++;        //Increment node degree
        node[e.tail].out_degree++;
        node[e.head].degree++;        //Both ends
        node[e.head].in_degree++;
        return true;
    }
    return false;                  //Not OK - duplicate exists
  }//NW_doAddLink
```

Gilbert 网络可能不是连通的。实际上，Gilbert 证明了通过他的过程可以生成以概率 $n(1-p)^{n-1}$ 包含多个组件并以概率 $2(1-p)^{n-1}$ 隔离节点（度等于零）的随机网络。尽管对于 $n \gg 1$ 的大网络来讲，这些是非常小的概率，仅仅隔离节点的可能性就使得这种方法不适用于很多应用。

除了有关隔离节点数的不确定性外，Gilbert 网络不能保证包含准确数目的链路。例如，如果 $p = 50\%$，$n = 100$，那么平均 Gilbert 网络将包含 $m = 0.5\left(n\frac{n-1}{2}\right) = 0.25(100(99)) = 2475$ 条链路，但是典型 Gilbert 网络一次可能包含 $m = 2470$ 条链路，另外一次包含 2480 条链路，第三次包含 2479 条链路。因为插入链路是一个随机过程，在最终的网络中链路数目是不确定的和不可预测的。这或许是这种方法的最重要的不足之处。

在最终对 m 值缺乏预测性使得 Gilbert 网络成为历史。大多数随机网络是由当今的 Erdos-Renyi（ER）算法生成的。ER 网络能保证刚好有 m 条链路，服从二项式度序列分布，并且很容易被生成。但是，ER 网络可能是不连通的。

4.1.2 Erdos-Renyi 随机网络

更好的随机网络生成过程是 1959 年由 Paul Erdos 和 Alfred Renyi 所建议的，也就是 ER 生成过程。这是目前产生随机网络的标准方法。与 Gilbert 过程不同，ER 生成过程固定链路数 m 和节点 n 并去掉了概率变量 p。但是像 Gilbert 网络一样，它很可能会生成一个带有孤立组件和孤立节

⊖ 我们在不同的程序中使用该方法的不同版本，因此 NW_doAddLink()在不同的章中会有所不同。

点的 ER 网络。之所以如此是因为每个节点对是随机选择的，这就意味着很可能根本不选择某一个节点。小心实现 ER 生成过程，可以避免循环和链路重复，但是不能保证生成一个强连通的网络。

给定 n 和 m，构建一个 ER 随机网络如下：

1. 初始化：生成 n 个节点并从 0 到（$n-1$）进行编号。
2. 初始化：给定 m，并且#links（链路数）=0。
3. 重复以下过程，直到 m =已经插入的链路数为止：
   a. 选择随机的节点：tail =（Math. random()）n。
   b. 选择随机的节点：head =（Math. random()）n。
   c. 避免循环：while(tail = = head)head =（Math. random()）n。
   d. 避免重复：if（没有重复）就在 tail（尾）和 head（头）之间插入一条新的链路，并对#links 增 1。否则，什么都不做。

这种生成过程避免了循环，因为步骤 3c 保证 tail 和 head 节点是不同的。由于步骤 3d，该生成过程可以避免重复。该步骤集成到前面描述的 NW_doAddLink( )方法之中。尝试过 m 条链路后，ER 过程停止。因此，m 必须小于 $n((n-1)/2)$，否则过程就永远不会停止！可能出现少于 m 条的插入链路，因为当可能导致重复链路时就没有插入链路。

ER 网络具有特定数目的链路，因此对于给定节点和链路数，其密度为 $2m/(n(n-1))$。这就允许更精确地控制随机网络的密度。例如，如果 $n=100$，$m=200$，由 ER 过程生成的每一个随机网络的密度为：

$$\text{density} = \frac{\frac{m}{\frac{n(n-1)}{2}}}{} = \frac{200}{50(99)} = 0.0404$$

ER 和 Gilbert 过程都生成一种随机网络，因为两种情况下度序列分布都逐渐地接近泊松分布。随机网络生成过程与泊松过程之间的关系稍后介绍。结果，当节点数和链路数相等时熵和聚类系数是可以比较的。但是 Gilbert 过程产生一个具有某一密度的随机网络，而 ER 过程产生一个具有某一数目链路的随机网络。尽管这似乎是一样的，但 ER 过程会固定链路数而 Gilbert 过程则不会。

ER 过程的 Java 代码就像上述建议的微规则那样简单。方法 NW_doCreateER( )首先创建 n 个节点，然后方法 NW_doCreateERLinks( )在 m 个随机选择的节点对之间插入 m 条链路。重复链路由方法 NW_doAddLink( )处理，用于检测保证节点对之间不存在链路。当插入一条新的链路时，这种方法也将 nLinks 增 1。因此，while 循环最终中止，因为 nLinks 将最终增加到 nInputLinks 值，它将等于上面的 m：

```
public void NW_doCreateER() {
       NW_doCreateNodes(nInputNodes);     //Create nodes, only
       NW_doCreateERLinks();              //Create links, randomly
}//NW_doCreateER
 private void NW_doCreateERLinks() {
    int max_links = nNodes * (nNodes-1)/2;
    nLinks = 0;
    if(nInputLinks > max_links) return;
    int to, from;                 //Randomly selected node-pair
    while(nLinks > nInputLinks){    //Don't stop until all links are created
     from = (int)(Math.random()*nNodes);
     to = (int)(Math.random()*nNodes);
     while(from == to) to = (int)(Math.random()*nNodes); //Cannot connect to self
     NW_doAddLink(node[from].name, node[to].name, Long.toString(nLinks));
 }
}//NW_doCreateERLinks
```

4.1.3 锚定随机网络

Gilbert 和 ER 网络大多数情况下生成带有单个组件的随机网络，但是并非总是如此。对 ER 生成过程稍加修改，通过保证每个节点至少有一条链路，就能保证所有的节点连接到至少一个其他节点上。这通过访问每一个节点至少一次来完成（以循环的方式），并测试每个节点的度。如果度为零，算法将链路的尾节点连接到孤节点上；否则就随机地选择尾节点。

结果锚定随机网络必须至少有 $m = n/2$ 条链路，因此每个节点至少有一条链路连接。通过所有节点的第 1 趟是系统地进行的——它从节点 0 开始，并按次序 $0,1,2,3,\cdots,(n-1)$ 访问每个节点。如果 $m > (n/2)$，下一个和随后的一趟通过索引 $0,1,2,3,\cdots,(n-1)$ 进行，但是要随机地选择节点，而非同一顺序的节点。由于首次系统地按序通过所有节点而导入了稍小的偏差，结果网络要比随机的网络稍小。

给定 n 和 m，构建一个锚定随机网络如下：

1. 初始化：生成 n 个节点并从 0 到 $(n-1)$ 进行编号。
2. 初始化：给定 $m \geqslant (n/2)$，并且 #links = 0。
3. 重复以下过程，直到已经插入的链路数 m = #links 为止：
   a. 循环：$i = 0,1,2,\cdots,(n-1);0,1,2,\cdots$。
   b. 选择尾部：If (degree(i) >0) tail = (Math. random()n, else tail = i。
   c. 选择随机节点：head = (Math. random())n。
   d. 避免循环：While (tail = =head) head = (Math. random())n。
   e. 避免重复：If（没有重复），在 tail 和 head 之间插入新的链路并对#links 增 1。否则，什么都不做。

对应于这种生成过程的 Java 方法给出如下。这会比前面的生成过程更为复杂，这是因为前面刚刚介绍过的步骤 3。但是，这与纯 ER 过程非常相似，因为它生成规定链路数目的随机网络。如果链路太少，那么 NW_doCreateAnchored()将数目提高到 $m = (n/2)$。相反，如果链路太多，则方法退出。总的来说，链路要比节点多，因此方法以循环的方式 $0,1,2,\cdots,(n-1),0,1,2,\cdots$ 访问每个节点几次：

```
public void NW_doCreateAnchored() {
    NW_doCreateNodes(nInputNodes);      //Create nodes, only
    int count = nInputLinks;            //Number links >= n/2?
    if(count  < nInputNodes/2) count = nNodes/2;
    if(count \gt= nNodes*(nNodes-1))  return; //Too many links
    int tail = 0;               //Starting node
    int i = 0;                  //i = 0, 1, 2...(n-1), 0, 1, 2...
    while(nLinks  < count){
        if(node[ tail] .degree > 0)
        i = (int)(Math.random() * nNodes);
        else i = tail;
        int head = (int)(Math.random() * nNodes);
        while(head == tail) head = (int)(Math.random() * nNodes);
        if(NW_doAddLink(node[ i] .name, node[ head] .name)) {
            tail++;
            if(tail == nNodes) tail = 0; //Wrap around (n-1) -> 0
        }
    }
}//NW_doCreateAnchored
```

度序列分布有点偏斜，因为当 $m \geqslant (n/2)$ 时所有节点具有非零度。当熵稍低于纯随机网络时，这种偏差就显示出来。回顾度序列分布服从二项式分布，节点无链接的概率由非零的 $B(0, n)$ 给定。在熵方程中的该项不能被忽略。

尽管锚定随机网络过程保证所有节点连接到某处，连接可能将分离的节点子集链接起来。

换句话讲，锚定随机网络可仍旧包含单独的组件，因为一个不连接节点子集可能仅链接到子集内的节点，使得其余的节点从子集出发就不可达。尽管这种情况非常少，但存在可能性。我们如何克服这种异常？该问题留给读者作为练习。

4.2 随机网络的度分布

随机网络是很多将在随后的章中要讨论的涌现过程的起点。总的来讲，我们从随机网络开始，然后观察它是如何随着时间演变而具有某种形式的非随机性的。这种行为（一种网络从随机性演变成有序）是涌现的基础。例如，随机网络是新产品市场的合适模型，用节点代表产品和消费者，产品和消费者之间的链路代表产品的购买（及其竞争者），随着市场的“形成”，随机网络变成一种由少量节点（产品）连接到很多其他节点（消费者）所主宰的无标度或小世界网络，希望随着消费者学习到新的产品，他们就会被吸引到这个网络上来。这种交换被建模成重联链路——从一组节点断开并连接到新产品的节点上。如此一来，代表一个新的市场的随机网络演变成一个表示成熟市场的结构化网络。换句话来讲，结构化网络从随机网络涌现，对应于从一个非结构化市场涌现为一个成熟市场。

PC 机市场是从混沌或随机新市场涌现的成熟的、结构化市场的经典例子。在 20 世纪 80 年代初期，PC 机市场上有数百公司，他们提供不同类型的 PC 机。到了 20 世纪 90 年代中期，早期 PC 机市场已经成熟是由少数主要市场领导者组成的结构化垄断。市场不再是随机的。相反，它是由少数高度连接节点表示的市场领导者。

涌现是重联或重构网络的过程。我们可以将此看做网络属性随时间而变化。最感兴趣的典型属性是度序列分布、平均路径长度、中心性和聚类系数。度序列分布告诉我们具有什么类型的网络拓扑——结构化的或随机的。平均路径长度和聚类系数可以告诉我们结构化网络是无标度的或是小世界的。中心性测量一个节点或少量中心节点对网络的影响。我们从检查随机网络的影响开始，并显示任何随机网络的序列分布是二项式分布的。

为了证明 Gilbert 和 ER 生成过程都服从泊松分布（$n \gg 1$），使用以下策略：（1）导出建模 ER 网络节点对选择过程的二项式分布；（2）对于非常大的 m 来讲，证明二项式分布渐近等于泊松分布。两种分布表示度为 k 的节点的概率，但是泊松分布从公式中去掉了 m（链路数）。因此，当讨论大网络时就会优先选择它。导出策略如下所示：

1. 证明节点对的随机选择服从二项式分布。
2. 证明随着链路数 m 增大，二项式分布变成泊松分布，如此一来从分布方程中去掉了 m。
3. 注意：应用当 m 无限增大时 $((1-\lambda)/m)$ 就变成 $e^{-\lambda}$ 的事实。

使网络 $G = [N(t), L(t), f(t)]$ 成为随机生成的 n 个节点 m 条链路的网络。进一步来讲，让 λ 表示从度序列分布 $g' = [h_1, h_2, \cdots, h_{max\_d}]$ 中获得的平均度值，这里 h_{max_d} 是 G 中最高的度。在 Gilbert 和 ER 两种生成过程中，$N(t)$ 是在时间 $t = 0$ 时创建的，并仍旧保持不变。$L(t)$ 从 $t = 0$ 时的零条链路开始，在生成网络时每时间单位以概率 λ/m 添加一条链路。在时间 $t = m$，按照 Gilbert 和 ER 过程添加了所有链路，因此 $L(t)$ 和 $f(t)$ 对于接下来的 $t > m$ 值保持不变。因此，G 的最后状态在 $t = m$ 时达到：$G = [N(m), L(m), f(m)]$。

在策略的第 1 部分，我们演示了 G 的度分布服从二项式分布。考虑 $n = 10$ 个节点、$m = 30$ 条链路的 G，因此 $avg\_degree(G) = \lambda = (2m/n)$ 条链路。按照 ER 生成过程，每个节点在 $m = 30$ 时间步平均接收 6 条连接，因为在每一时间步将一对节点链接起来。现在如图 4-1 中所示，重点在于节点 $v \in N$。该节点以概率 $p = (\lambda/m)$，在 m 时间步以概率 $p^k = (\lambda/m)^k$ 出现 k 次。它不被选择的概率也为 $(1-p)^{m-k} = \left(\frac{1-\lambda}{m}\right)^{m-k}$。

因此，节点 v 刚好选择 k 次的概率为这些概率的乘积

$$\text{Prob}(v \text{ 在 } m \text{ 步选择 } k \text{ 次}) = \left(\frac{\lambda}{m}\right)^k \left(\frac{1-\lambda}{m}\right)^{m-k}$$

但是 m 连接中的 k 次是以 $C\binom{m}{k}$ 种方式发生的。例如在图4-1中，链路1、2和30连接到 v，但是链路1，2，3或2，3，5或3，4，30等也可以相同的概率选择。因此，在 m 时间单位中成功选择 k 次的概率由二项式分布 $B(m,k)$ 给出：

$$\text{Prob}(\text{degree}(v) = k) = B(m,k) = C\binom{m}{k}\left(\frac{\lambda}{m}\right)^k \left(\frac{1-\lambda}{m}\right)^{m-k}$$

例如在图4-1中，节点 v 被选择 $k=3$ 次的概率为 $(\lambda/m)^k = \left(\frac{6}{30}\right)^3 = 0.008$，而三次不被选择的概率为 $((1-6)/30)^{30-3} = 0.00242$。节点 v 被选择 $k=3$ 次有 $C\binom{m}{k} = 30!/(3!(27!)) = 4060$ 种方式，因此 $\text{degree}(v)=3$ 的概率是乘积 $(4060)(0.008)(0.00242)=0.0785$。换句话来讲，节点 v 被 $k=3$ 条链路连接的概率是7.85%，因此度等于3。

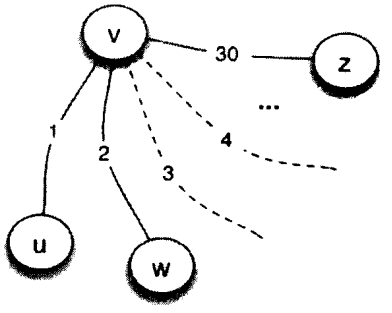


图4-1 $n=10$、$m=30$、$\text{avg\_degree}(G)=\lambda=3$ 时随机图 G 的节点 v。点画线指示还没有分配给 v 的链路，但是如果分配的话，$\text{degree}(v)$ 将从3提高到5

度序列分布是通过简单估计 $k=0,1,2,\cdots(n-1)$ 的二项式分布计算出来。$B(m,k)$ 从接近于零提高到最大值，然后再次下降到接近零。例如，$B(30,0)=0.00124$，$B(30,3)=0.0785$，$B(30,29)=0$。分布上升到它的峰值，然后缓慢地降低到接近于零。在图4-2中显示了由 Gilbert 和 ER 算法生成的随机网络的度序列分布的例子。

在导出策略的第2部分，注意随着 n 和 m 的增加，$B(m,k)$ 接近于泊松分布。链路数 m 要比节点数增加得更快。通过保持 n 不变而让 m 无限制地在 $B(m,k)$ 中增加来建模这种关系：

$$\lim_{m\to\infty}\{B(m,k)\} = \lim_{m\to\infty}(m(m-1))\cdots\left(\frac{m-k+1}{m^k}\right)\left(\frac{\lambda^k}{k!}\right)\left(1-\frac{\lambda}{m}\right)^m$$

在 $B(m,k)$ 形式中，在 $C\binom{m}{k}$ 分母项的 $k!$ 与 $(\lambda/m)^k$ 中的项 m^k 互换，以备取极限。这种互换之后，$B(m,k)$ 可以写成三项 T_1、T_2、T_3 的乘积：

$$T_1: m(m-1)\cdots\frac{(m-k+1)}{m^k}$$

$$T_2: \frac{\lambda^k}{k!}$$

$$T_3: \left(\frac{1-\lambda}{m}\right)^m$$

$$B(m,k) = (T_1)(T_2)(T_3)$$

接下来让 m 无限地增加。随着 m 无限制地增加，T_1 接近于1，因为它是在分子和分母中都包含 m 项的乘积：$O(m/m)$。既然 T_2 不是 m 的函数，它就不会改变。T_3 是一种特殊例子——随着 m 无限制地增加，它接近于 $e^{-\lambda}$，因为 λ/m 要比 $((1-\lambda)/m)^m$ 的增加稍微更慢地消失，其证明留

给读者完成[⊖]。

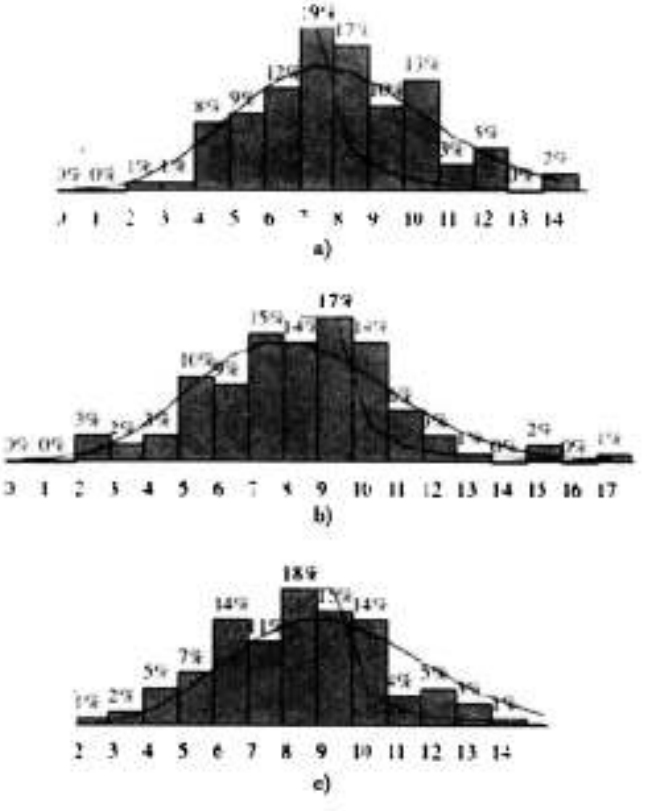


图4-2 由不同生成过程生成的随机网络（$n=100$，$m=400$）的度序列分布：a）Gilbert生成过程；b）ER生成过程；c）锚定ER生成过程

将上述这些分析综合起来，我们就得到：

$$\lim(m\to\infty)T_1=1,\ \lim(m\to\infty)T_2=\frac{\lambda^k}{k!}$$

$$\lim(m\to\infty)T_3=e^{-\lambda},\ \lim(m\to\infty)B(m,k)=\frac{\lambda^k e^{-\lambda}}{k!}$$

这就是泊松分布$P(\lambda,k)$，其中$k=0,1,\cdots$，对于大的随机网络来讲$(n,m\gg1)$，它代表度序列分布。Gilbert和ER网络服从这种定律，因为重复地选择过程是均匀随机的。我们之所以称生成过程为泊松分布就是因为这种原因。许多随机过程是泊松分布过程，是因为一致的单个事件发生很多次，生成了通过泊松方程建模的多个事件。

图4-2显示运行程序Network.jar的结果，其中$n=100$，$m=400$，连接链路概率为$p=8\%$（Gilbert）。水平轴标示节点度d，垂直轴标示度为d的节点百分数。Gilbert和ER分布很相似，但是锚定网络分布稍稍偏向左，并具有稍小的方差。这种偏差是锚定ER生成过程的结果，它故意地将所有节点链接到至少一个其他节点上。

在柱状图数据上Network.jar覆盖了泊松分布/幂律分布曲线，因此便于看出是否匹配。这些显示为叠加在柱状图上的细线图。显然，实验数据（柱状图）很好地匹配了理论泊松分布线图，但是不能匹配幂律分布。这就部分地证实了Gilbert和ER网络是随机的，锚定ER类网络是接近

⊖ 提示：将$((1-\lambda)/m)$展开成围绕$x=1$的泰勒级数，并注意泰勒级数展开等同于$\exp(-\lambda)$的泰勒级数展开。

随机的。

4.3 随机网络的熵

比起本书中研究的任何其他网络，随机网络的熵要高得多。在第2章中，我们演示了随机网络类位于结构化频谱的一端，而 k - 规则网络位于另一端。但是，可能不那么明确的就是随机网络变得越密就更加结构化！详细讲来，随着链路数接近最大值 $n((n-1)/2)$，$m \gg 1$，熵快速下降。这种惊人的结果可以通过实验加以演示。

图4-3显示了这种实验结果。程序 Network. jar 对于 $n = 100$、$m = 100,200,\cdots,4900$ 运行很多次。密度为 $2m/(n(n-1))$，因此我们画出熵 I 和密度之间的关系图。对于每个密度值生成了5个 ER 网络，并平均了5个熵值。这种平均形成的数据显示在图4-3中。

熵（和随机性）随着密度的增加而快速增加、趋平，然后随着网络密度接近100%再次下降。在图4-3中显示的多项式并非准确地在50%处对称，但它是随机网络的“随机量”从零增加到最大值在接近 $m = n((n-1)/4)$（density = 50%）处形成的明确的曲线，然后在 $m = n((n-1)/2)$（density = 100%）处减少到零。换句话来讲，随机网络仅在密度值范围的中间才是完全随机的。ER 随机网络过程仅为 $m = n((n-1)/4)$ 条链路生成一个真正的随机网络。

随机网络的随机性随着密度接近100%而减少，因为此时网络拓扑变得非常规则了。我们知道完全网络是规则网络，因为所有可能的链路都出现在网络中。完全网络的熵为零。随着随机网络变得更稠密，每个节点的度接近完全网络的度。按数学术语来讲，随机网络中节点的度随着 m 接近于 $n((n-1)/2)$ 而接近于 $(n-1)$。度序列分布为 $g' = [(n-1)]$，这与泊松分布相差很远！另外一种解释就是随着密度偏离50%，随机网络就不那么随机了！

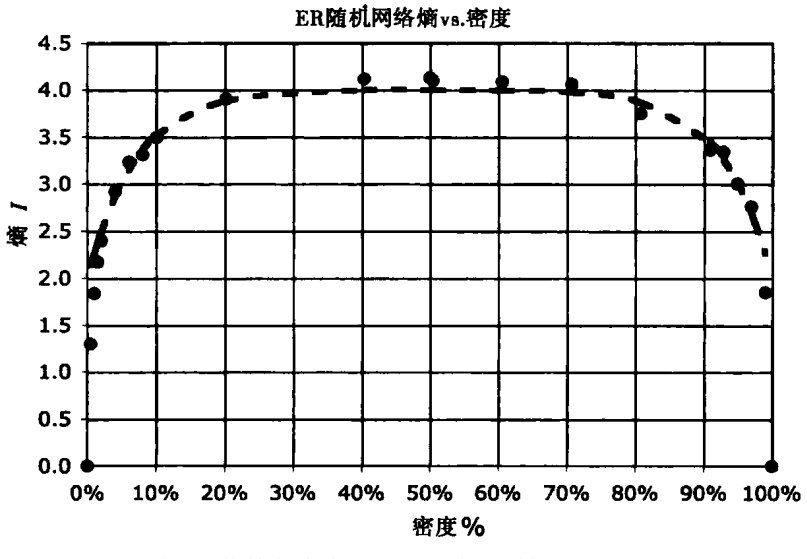


图4-3 ER 随机网络熵与密度的对比（密度看做完全连接（完全）网络的百分比：$n = 100$，$m = 100 \sim 4950$）

4.3.1 随机网络熵的建模

我们使用图4-3导出随机网络熵 I（random）的密度函数表达式。熵也是链路数 m 的函数，因为密度 = $2m/(n(n-1))$。因此我们可以使用其中任意一个，但是密度是一种方便的度量。

下面的推导是根据熵在50%处是对称的而进行的，但是这仅是一种近似。也要注意到图4-3中的数据不平滑——它们并没有完全地落在这里导出的平滑曲线上。以下策略产生 $n = 100$ 的随机网络熵的近似：

1. 注意图4-3在大约50%密度处接近对称，熵以指数增长直到接近4.0为止，然后呈指数下降直到返回零为止。

2. 估计图4-3左半部分（0% ~50%）的熵与密度，然后“翻转”x轴获得右半部分（50% ~100%）熵的镜像表示。

3. 将图4-3的两个部分结合起来：$I(\text{random}) = 0.5$（左半部分 + 右半部分）。

4. 将合成表达式简化成一个方程。

在图4-3的左半部分，熵呈指数上升然后在接近密度=50%时变平，并且相似地经过图4-3的右半部分降低到零。这样一来，左半部分和右半部分的表达式建模成$I(x)$和$I(1-x)$,$0 < x < 1$：

$$\text{left}(x) = A(1 - \exp(-Bx)); \quad \text{左半部分}$$

$$\text{right}(x) = A(1 - \exp(-B(1-x))); \quad \text{右半部分}$$

将两个半部分结合起来，就得到：

$$I(x) = 0.5[\text{left}(x) + \text{right}(x)] = 0.5[A(1 - \exp(-Bx)) + A(1 - \exp(-B(1-x)))]$$

最小二乘曲线拟合产生如下对A和B的估计：

$$A = 4, B = 13; \text{假定 } n = 100$$

代入到$I(x)$表达式中并简化，就得到：

$$I(x) = 4 - 2[\exp(-13x) + \exp(-13(1-x))]; \quad 0 < x < 1$$

因此，图4-3中随机网络的熵近似为：

$$I(\text{random}) = 4 - 2[\exp(-13(\text{density})) + \exp(-13(1-\text{density}))]; \quad 0 < \text{density} < 1$$

这种近似的均方根（RMS）误差为0.0545，因此这就产生熵的估计为每100个中误差在5个之内。最大的误差发生在密度的极端值，分别位于接近0%和100%。例如，密度为1%的随机网络的熵为1.83，但是近似生成2.25。对于$0\% \ll \text{density} \ll 100\%$，近似很快提高。对于密度10%，近似数据和实验数据两者都为3.46。这种近似适用于中等密度值。

4.3.2 随机网络的平均路径长度

随机网络的平均路径长度应该会随着链路数目的增加而减少，由于节点对之间的路径数扩散——这样为更短的路径提供更多机会。如图4-4所示，这刚好与实际相符。当ER和Gilbert随机网络的平均路径长度在重对数（log-log）尺度画出时，随着密度增加到100%，平均路径长度急剧降低。实际上，平均路径长度在密度达到100%时应该渐近达到1跳，因为完全连接网络会将每一个节点连接到其他节点上。图4-4中的曲线如预测的那样渐近降低到$\log_2(100\%) = 0$。

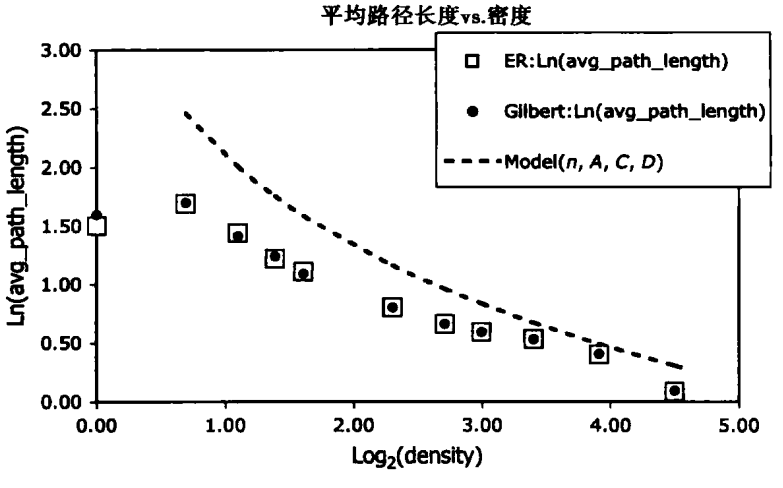


图4-4 ER网络、Gilbert网络平均路径长度与链路密度对比以及根据修改过的理论近似和底为2的对数建模（点画线）：$n = 100$，$A = 1.32$，$C = 1.51$，$D = 0$

Newman（Newman, 2000b; Albert, 2002）导出了平均节点度为 λ 的随机网络中节点间距离的近似，假定沿着从任意节点 v 到另外一个节点 w 路径到达的节点数接近 $\lambda^D = n$。Newman 讨论了经过 1 跳就可以到达的节点有 λ 个，经过两跳就可以到达的节点有 λ^2 个，经过 3 跳可以到达的节点有 λ^3 个等，直到以 λ^D 跳就可以到达的节点有 n 个为止。求解 D，我们就得到

$$D = \frac{\log(n)}{\log(\lambda)}$$

均匀的随机网络的平均路径长度与 D 成比例，因此 D 是对平均路径长度的很好近似。但是 D 仅是在假定网络为每个节点连接到其他 λ 个节点上的环形树时的一个近似。随着节点密度在两个方向到达极端，或许是因为网络拓扑对于稀疏网络不那么均匀，而对于密集网络不那么随机造成的，那么假定会导致更不精确。

注意到以下事实可获得稍微更精确的近似：在同一强连通子图中从任意节点 w 到所有其他节点的路径，以 1 跳到达 λ 个节点，以 2 跳到达 $\lambda(\lambda-1)$ 个节点，因为邻居的邻居有 $(\lambda-1)$ 条外出链路不会回溯到进入的链路（用于以宽度优先搜索到达第二层节点）。在扩展宽度优先搜索中随后以 D 跳到达 $\lambda(\lambda-1)^{D-1}$ 个节点。如果整个网络属于一个大的强连通的子图，那么所有 n 个节点都以 D 跳到达：

$$n = \lambda(\lambda-1)^{D-1}$$

通过求解 D 来近似平均路径长度，我们就得到：

$$D = \frac{\log(n/\lambda)}{\log(\lambda-1)+1}$$

假设我们通过标记 $\lambda = n(\text{density})$，并导入曲线拟合参数 A、C 和 D，总结出理论近似：

$$\text{Model}(n,A,C,D) = \frac{A\log(n)}{\log(n(\text{density})C)+D}$$

这个变量是对固定 n 和可变密度的路径长度的很好近似，如图 4-4 中最小二乘拟合所示。对于底为 2 的对数，$A = 1.32$、$C = 1.51$ 和 $D = 0$，因此平均路径长度近似地为

$$\text{avg\_ path\_length}(\text{random, denstiy}) = \frac{A\log_2(n)}{\log_2(n(\text{density})C)} = \frac{1.32(\log_2(n))}{\log_2(n(\text{density})1.51)}$$

考虑一个规模为 $n = 100$、链路数 $m = 500$ 的 ER 随机网络。假定参数从图 4-4 中获取，那么平均路径长度为多少？

$$\text{density} = \frac{2m}{n(n-1)} = \frac{1000}{9900} = 0.101 = 10.1\%$$

$$\text{avg\_ path\_length}(\text{random}, 0.101) = \frac{1.32\log_2(100)}{\log_2((100)(0.101)(1.51))} = 1.32\left(\frac{6.64}{3.93}\right) = \frac{8.77}{3.93} = 2.23 \text{ 跳}$$

模型总的来讲过高估计了平均路径长度，如图 4-4 所示，但是对于中等密度非常准确。在这种情况下，当 $n = 100$、密度 = 10% 时 Gilbert 和 ER 生成过程产生平均路径长度为 2.23 跳的网络。近似数据与经验数据相一致。

在图 4-4 中，注意平均路径长度在密度等于 1% ~4% 区间快速减少，然后超过 4% 后就以非常慢的速度降低。这是规则网络、随机网络和小世界网络的一个重要属性。添加少量的链路后减少的路径长度要远远低于预期。例如，在 $n = 100$ 个节点的 2 - 规则网络中添加 5 条随机链路，平均路径长度会从 12.9 降低到 7.1 跳。仅添加了 2.5% 的链路，平均路径长度就降低了 45%。

添加少量随机链路会导致平均路径长度快速降低，这称为*小世界效应*。随机网络会显示小世界效应，就像小世界网络一样。为什么？总的来讲，添加少数随机链路创建了将（近似）一半的网络链接到另外一半的穿越网络的捷径。每次连续添加随机链路都会对分网络，将任意选

择的两个节点之间距离减少 $\frac{1}{2}$、$\frac{1}{4}$、$\frac{1}{16}$ 等。小世界效应将在下一章中详细研究。

4.3.3 随机网络的聚类系数

随机网络中链路密度的增加会导致平均路径长度的减少，而聚类系数则刚好与此相反，却会增加。聚类系数随着密度的增加而增加（参见图 4-5）。实际上，它是呈线性增加的。一个聚类系数与随机网络密度的简单模型为

$$\text{cluster\_coefficient(random)} = O(\text{density}); \quad 1\% \leqslant \text{density} \leqslant 100\%$$

聚类系数会随着密度增加而增加是因为更多链路就更容易形成三角子图。当密度等于 100% 时，网络就不再随机了——而是变成完全的。在完全网络中，每个节点连接到其他节点上，因此每个节点的聚类系数为 1.0。换句话来讲，随着网络变得稀疏，它的聚类系数降低——最后变为零。

Watts 和 Strogatz（Watts，1998）导出了随机网络聚类系数的理论估计：

$$\text{cluster\_coefficient(random)} = \frac{\lambda}{n}$$

这里 λ = 平均度 = $(2m/n)$ = $(n-1)$ 密度 ~ n(密度)；$n \gg 1$。带入到 Watts-Strogatz 方程中产生一个简单的、被图 4-5 实验结果证实的关系：

$$\text{cluster\_coefficient(random)} = \text{density} = O(\text{density})$$

在随机网络中链路越多，聚类就越多。回顾聚类是由于链接相邻邻居形成三角子图时创建的。附加到节点的三角子图越多，聚类系数就越高。提高密度会提高三角子图形成的相似性。

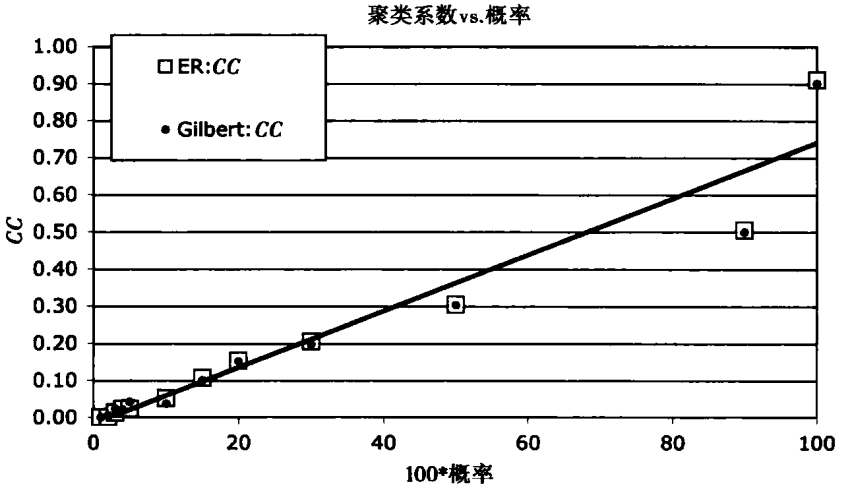


图 4-5 ER 和 Gilbert 随机网络的聚类系数与链路密度（概率）。直线近似 cluster_coefficient(random) = O(density) 的关系

根据定义，平均度为 λ 的节点 v 的平均聚类系数为：

$$CC = \frac{2c}{\lambda(\lambda-1)}$$

这里 c 为邻接节点的链路数量。再次，使用平均场方法估计，相邻节点的链路数（形成三角子图）为 λ 个中每次取出两个：

$$C\binom{\lambda}{2} = \frac{\lambda!}{2!(\lambda-2)!}$$，如果网络完全连通；

$$\text{density}\left(C\binom{\lambda}{2}\right) = \text{density}\left(\frac{\lambda!}{2!(\lambda-2)!}\right)$$

如果网络是稀疏的。那么在聚合中给出的链路比例等于网络的总密度。因此，c 的平均场近似为

$$c = \text{density}\frac{\lambda!}{2!(\lambda-2)!} = \lambda(\lambda-1)\frac{\text{density}}{2}$$

带入到 CC 的表达式中就得到：

$$CC = \lambda(\lambda-1)\frac{\text{density}}{\lambda(\lambda-1)} = \text{density} = \frac{\lambda}{n}$$

4.3.4 随机网络的链路效率

随机网络的链路效率很容易从前面进行的路径长度分析中计算出：

$$E(\text{random}) = \frac{m - \text{avg\_path\_length}(\text{random})}{m} = 1 - \frac{\text{avg\_path\_length}(\text{random})}{m}$$

链路数 m 和密度 d 通过 $d = \frac{2m}{n(n-1)}$ 关联起来，因此链路效率可以用 m 和 d 表示：

$$E(\text{random}) = 1 - \frac{A\log(n)}{m\log(n(\text{density})C)}$$

例如，当 $n = 100$，$d = 0.04$（$m = 200, A = 1.32, C = 1.51$）时：

$$E(\text{random}) = 1 - \frac{A\log(n)}{m\log(n(\text{density})C)} = 1 - 1.32\frac{6.64}{200\log(6.04)} = 1 - \frac{8.77}{(200)(2.59)} = 0.983$$

将该结果与先前从规则网络中获得的结果相比较（参见表 3-1）。由于小世界效应，随机网络能高效地利用链路。

因为在任何网络中的少量随机性注入会导致平均路径长度的很大降低，随机性会导致链路效率的很大跳跃！在实际中也可以观测到这种现象。例如，具有小世界效应的生物网络要比预期的更加容易同步，因为它们具有高水平的聚类。聚类由三角子图构成，结果证明这些三角子图可以保证同步——一种心脏起搏器和某种蟋蟀的属性。

4.4 随机网络的属性

社会网络分析主要在于社会网络中个体（节点）的有用性。但是在不同的分析中对“有用性”的定义是不同的。社会有用性是否与以下方面有关：有多少条链路将个体与其他人连接起来（hub 分析）？一个个体离其他个体（半径或中心性）有多远或者个体的中介（媒介）位置（紧度）是什么？一般来讲，选择的属性要根据社会科学家所问的问题来定。在像 Milgram 实验的研究中，直径或半径就足够了，中心性更适用于流行病学（人类或互联网结构中），介数/紧度可能更加适用于组的动态性描述。对人类网络理解的含义将在第 10 章中详细研究。

网络的直径是穿越所有节点对的最大长度路径，网络中心是所有从任意一个节点到其他任意节点的最大长度路径中的最小值。正如期望的那样，直径和中心性随着网络密度的增加而降低。此外像平均路径长度一样，直径的减少也是急剧的，如图 4-6 所示。再次，这是由于小世界效应引起的。一般来讲，直径为 1 跳，大于平均路径长度。令人惊讶的是，中心性可以大于平均路径长度。为什么？

直径和半径随着密度的增加骤减，但是紧度则不会！这种与直觉相反的结果表明：这是两种力量同时作用于随机网络的直接路径结构的结果——增加链路数也会增加通过典型节点的路径数，增加密度也会降低平均路径长度，并降低紧度！我们演示紧度快速增加到峰值，然后缓慢地减少到零，对于规模为 $n = 100, 200$ 的网络超过了大约 20% ~30%。

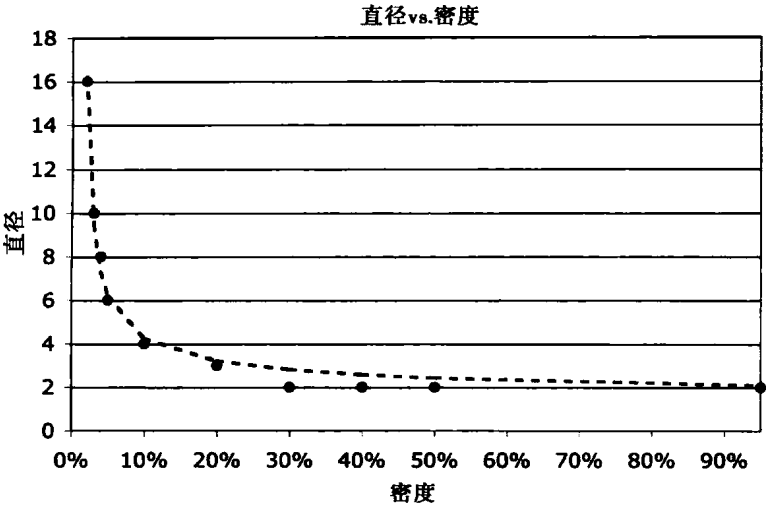


图4-6 $n = 100$ 的随机网络的直径与网络密度。点画线图是从近似参数 $A = 2.0$、$C = 0.44$、$D = 1.0$ 获得的

4.4.1 随机网络的直径

我们使用修改过的平均路径长度模型来对直径随着密度的增加而减少建模：

$$\text{Model}(n,A,C,D) = \frac{A\log(n)}{\log(n(\text{density})C) + D}$$

对于 $n = 100$，并对于每一个密度值运行3次 Network. jar 然后平均，就得到接近直径的参数[⊖]：

$$A = 2.0, C = 0.44, D = 1.0 \qquad \text{Diameter}(\text{random}) = \frac{2\log(n)}{\log(0.44n(\text{density})) + 1}$$

例如，假定密度 =0.50。比较 $n = 100$ 个节点的随机网络的直径和平均路径长度：

$$\text{直径} = \frac{2\log(100)}{\log((0.44)(100)(0.5)) + 1} = 2.43 \text{ 跳}$$

$$\text{平均路径长度} = \frac{1.32\log(100)}{\log((100)(0.5)(1.51))} = 1.41 \text{ 跳}$$

当然，跳数必须为整数，因此直径等于3跳，而平均路径长度等于2跳。

4.4.2 随机网络的半径

随机网络的平均路径长度和直径随着密度的增加而“缩减”。类似地，随着密度的增加而网络的中心性会“缩减”，按照如下半径的近似

$$\text{radius}(\text{random}) = \frac{A\log(n)}{\log(n(\text{density})C) + D}$$

对于 $n = 100$，$A = 1.59$，$C = 0.88$，$D = 0.5$。例如假定 $d(\text{density}) = 50\%$：

$$\text{radius}(\text{random}) = \frac{1.59\log(100)}{\log((100)(0.5)(0.88)) + 0.5} = 1.59\frac{6.64}{\log(44) + 0.5} = \frac{10.56}{5.46} = 1.77 \text{ 跳}$$

但是既然跳数为整数而非分数，那么就得到2跳。

为什么中心节点的半径大于平均路径长度（分别为1.77与1.41）？中心节点的半径不是最短路径的最小值，而是最长路径的最小值！换句话来讲，半径等于中心节点到最远节点的跳数。相

⊖ 在这些曲线拟合中使用底为2的对数。

反，平均路径是最短和最长路径的平均。因此，平均路径完全可能小于中心节点的半径。如果仅有一对节点相隔距离甚远，网络的半径就非常大。

4.4.3 利用 Java 计算紧度

节点 w 的紧度定义成从每一个其他节点 u 到每一个其他节点 v 通过节点 w 的有向路径数。我们必须跟踪从每个节点到每个其他节点的 n^2 条最短路径，计算沿着每一路径通过节点的路径数，然后将每一节点的计数加起来。这对计算机来说是一个非常费力的工作！下面显示的部分代码，初始化网络的节点然后调用方法 doShortestPath(start_node, end_node) n^2 次（对每对节点调用一次）。通过每个节点的路径总数在 node[i].value 中累计起来，标准化的紧度值将在 node[i].radius 中返回。

```
for(int start_node = 0; start_node < nNodes; start_node++){
        for(int end_node = 0; end_node < nNodes; end_node++){
            for(int i = 0; i  < nNodes; i++) {//Reset search flags
                node[i].visited = false;
                node[i].level = 0;
            }
            doShortestPath(start_node, end_node);
        }
    }
```

方法 doShortestPath() 使用与第 2 章中介绍过的同样的宽度优先搜索（BFS）技术。每个节点的层次与计算的 starting_node 有关，并存储在 node[i].level 中。但是与仅关心路径长度的平均路径长度计算不同，doShortestPath() 必须沿着最短路径回溯并且在离开时标记回溯过的节点。这部分算法复杂，因为有很多节点留在被访问过的队列中，但是没有与最短路径对应。正因为此，我们必须使用压栈记住链路和节点。当我们沿着链路和节点回溯时，去掉没有与属于最短路径的节点相连的链路，我们对访问次数进行计数，并将它们存储在 node[i].value 中。

算法有两个阶段：阶段 1，执行 BFS 以便找到目的地节点 end；阶段 2，沿着访问过的节点和链路回溯直到返回到起点开始为止。同一层次的节点存储在 FIFO 队列的 que 中，沿着搜索路径的链路存储在 FILO 栈 pds 中。去掉不与节点匹配的链路。这就保证找到唯一的路径，避免死路和循环：

```
private void doShortestPath(int start, int end) {
        if(start == end) return;
        boolean found = false;                  //break from loops
        int n0 = start;                         //Current node#
        int next_node = -1;                     //Spanning tree nodes
        int l0 = -1;                            //Current link#
        int n_level = 0;                        //Current level
        node[n0].level = 0;                     //Start here
        node[n0].visited = true;                //Avoid cycles
        Stack que = new Stack(nNodes+1);        //Node FIFO queue
        Stack pds = new Stack(nLinks+1);        //Link pushdown stack
        que.init();
        que.push(n0);                           //Remember nodes
        pds.init();                             //Remember links
        while(que.notEmpty() && !found){//Forward BFS
            n_level++;                          //Shortest path++
            n0 = que.pull();
            for(int e = 0; e  < nLinks; e++){
                next_node = -1;                 //Find link to next_node
```

```
            if(Link[ e] .head == n0 && !node[ Link[ e] .tail] .visited)
                next_node = Link[e].tail;
            else if(Link[ e] .tail == n0 && !node[ Link[ e] .head] .visited)
                        next_node = Link[e].head;
            if(next_node == -1) continue; //Skip node
            pds.push(e);
            found = (next_node == end);
            if(found) break;
            node[next_node].visited = true;
            node[next_node].level = n_level;
            que.push(next_node);          //Not found
        }//Over links
    }//Over nodes
    //Phase 2
    if(found){
        n0 = next_node;
        found = false;                    //Backtrack to start
        while(pds.notEmpty() && !found){
            l0 = pds.pop();               //Pop a link
            next_node = -1;               //Find next node
            if(n0 == Link[ l0] .head)     //Ignore all but...
                next_node = Link[l0].tail;
            else if(n0 == Link[ l0] .tail)
                next_node = Link[l0].head;
            if(next_node >= 0 && node[ next_node] .level  < n_level){ //Backtrack
                found = (next_node == start); //Start reached?
                if(found) break;
                node[next_node].value++;
                n0 = next_node;
                n_level-- ;
            }
        }
    }
    else Message = "Error: Path Not found!";
    return;
}//doShortestPath
```

这种方法在变量 node[i]. value 中返回通过每一个节点的路径数，但是这还不是我们想要的标准化的紧度。因此方法 NW_doCloseness( )必须标准化如下：

```
r.big = 0;
double avg_radius = 0.0;
if(max_closeness > 0){              //Normalize closeness
    r.big = 100;
    for(int i = 0; i  < nNodes; i++) {
        node[i].radius = (int)(100*node [ i] .value/max_closeness);
        avg_radius += node[i].radius;
    }
    avg_radius = avg_radius/nNodes;
}
```

程序 Network. jar 在窗口中显示了带节点的网络，这些节点包含了存储在 node[i]. radius 中的标准化的紧度值，并带有一个平均紧密度值标示。最有用的节点的紧度值为 100，稍次的节点的紧度值在 [0，100]。标准化的紧度值越高，节点就越有用，因为这就意味着在网络中必须有更多的路径遍历节点，以便从一个节点到另外一个节点。

紧度与中心性非常不同，正如我们这里所定义的。中心性是一个距离度量，而紧度则是一个连接性度量。紧的节点是一个连接器——一种使一个节点与另外一个节点连接起来的中间节点。中心性测量中心节点到达所有其他节点需要的跳数。问题在于，哪个属性更加重要？

4.4.4 随机网络中的紧度

直观地讲，我们期望平均紧度随着链路数（密度）的增加而增加，因为更多的链路就意味

着更多的路径。但是实际情况并非如此，如我们在图4-7中所见。最初，紧度非常快地上升，不过它到达峰值后又慢慢下降。事实上，$n = 200$ 要比 $n = 100$ 的上升和下降更加明显。为什么？

如果我们测试如下假设：紧度最初的提高是由于路径数的增加而增加的链路数。但是最后，链路的增加由于小世界效应而产生了更短的路径。这些较短的路径绕过（更大量的）可选的较长的路径。短路径会短路掉较长的路径，它会减少通过典型节点的路径数。在某些密度点，通过典型节点的路径数开始降低。随着密度的上升，紧度度量先增加然后再降低。

这种假设得到了图4-8的支持，该图中显示了通过最近节点的标准化路径数与平均路径长度之间的相互关系。很显然，紧度和路径长度之间存在某种线性关系[⊖]：

$$\frac{\text{通过中间节点的路径数}}{n} = O(\text{avg\_path\_length})$$

进一步来讲，紧度和密度存在反比的关系，因为

$$\text{avg\_path\_length} = O\left(\frac{\log(n)}{\log(\lambda)}\right), \lambda = n(\text{density})$$

$$\text{通过中间节点的路径数} = O\left(\frac{n\log(n)}{\log(n(\text{density}))}\right)$$

因此，典型节点的紧度应该最终随着密度的增加而减少。但是这并非完整的描述，因为提高密度也会导致更多的路径。为了解释图4-7，我们需要考虑两个因素：平均路径的长度和有向路径数。

假设我们使用以下平均场的方法估计通过典型节点的路径数。节点 v 平均有 λ 条链路与之连接。每条链路将 v 连接到接近 λ^r 个其他节点上，这里 r = 网络的直径。因此，通过节点 v 的路径数应该与 λ^r 成比例，但是我们从图4-7可以知道，增加密度会导致围绕节点 v 的捷径。相反，按照 $(1 - \text{density})$ 通过节点 v 的路径数随着密度提高而降低。将这两个因素结合起来，我们得到：

$$100(\text{closeness}(\text{random})) = C_0(1 - \text{density})\lambda^r + C_1, \text{ 其中 } r = \frac{A\log_2(n)}{\log_2(B\lambda) + C}$$

这里 C_0、C_1、A、B、C 为常数。以下参数为近似100（平均紧度）产生图4-7中的实线曲线：

$$n = 100: C_0 = 0.2, C_1 = 22, A = 1.275, B = 1, C = 1.275$$

$$n = 200: C_0 = 0.21, C_1 = 1, A = 1.275, B = 1, C = 1.275$$

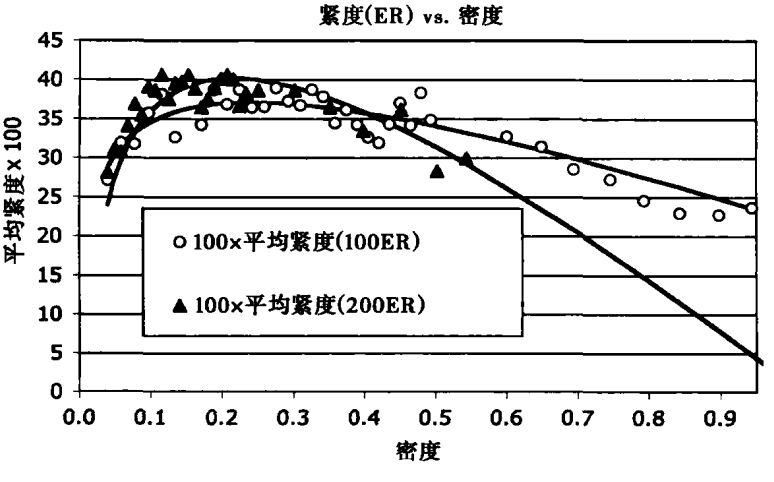


图4-7 规模为 $n = 100,200$ 的随机网络的平均紧度与密度。随着链路数的增加紧度上升到峰值然后下降

⊖ 有人会认为，由于“波浪式”的数据，图4-8中所示的关系并不完全遵守线性衰退。

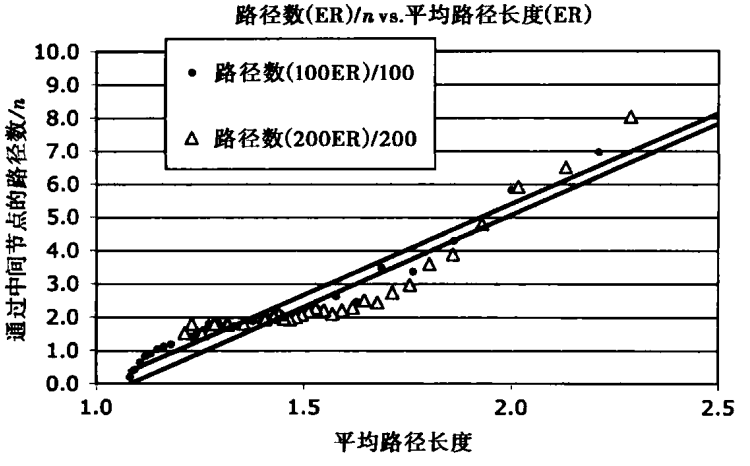


图 4-8 在规模为 $n = 100,200$ 的随机网络中，通过最大中间节点的路径数与平均路径长度

因此，closeness(random) $= O((1 - \text{density})\lambda')$，这就支持平均紧度为稀疏性（1 - density）与平均路径长度的函数的猜想。换句话来讲，紧度随着链路数的增加而增加，但是由于小世界效应很快就被最短路径长度的快速下降所淹没。正像我们将在随后的章节中所见，无标度网络或小世界网络却不会出现的这种情况。紧度是在不同类型的网络中表现不同的一种属性。

4.5 随机网络中的弱联系

考虑如下的社会网络例子。Flatland Pointville 的人口为 129。这个村子的不同之处在于每个人都同样地想要了解其他人，但是因为他们忙碌的社会生活，Flatland 的每个人只能与这里的 24 个其他人交朋友。我们想要预测如果在 Pointville 进行 Milgram 实验最有可能会发生什么？Pointville 市民间的隔离程度是多少？同样在 Pointville 中密切意味着什么？

通过 Pointville 的最长的和最短的弱联系是什么？回顾*弱联系*是这样一串熟人，即对于任何节点对 $u \sim w$，他们都能穿越整个人群而引导从 u 到 w。假设在 Pointville 中没有隐士，最长的和最短的弱联系等于这种人群的直径和半径。进一步，给定 Pointville 的平均路径长度，我们就能估计村民的平均紧度了。

解决该问题的策略是：

1. 从平均节点度 λ 导出链路数 m 的方程。
2. 将 m 的方程代入到密度中，并使用密度和 λ 计算直径、半径和平均紧度。
3. 假设 $n = 129$，平均节点度 $\lambda = 24$：使用从图 4-7 中导出的近似，并且 $n = 100$。

链路数 $m = \lambda(n/2)$，因为所有节点度总和等于两倍的链路数：$n\lambda = 2m$。将这些代入到网络密度的方程中：

$$\text{density} = d = \frac{2m}{n(n-1)} = \frac{2\lambda(n/2)}{n(n-1)} = \frac{\lambda}{n-1}$$

将密度代入到前面导出的直径和半径近似中（我们假定 $n = 100$ 和 $n = 129$ 给出相似的参数值 A、C、D）：

$$\text{diameter} = \frac{2\log(n)}{\log(0.44nd) + 1}$$

$$\text{radius} = \frac{1.59\log(n)}{\log(nd0.88) + 0.5}$$

接下来，令 $n = 129$，$\lambda = 24$：

$$d = \frac{24}{128} = 0.1875$$

$$\text{diameter} = \frac{2\log(129)}{\log((0.44)(129)(0.1875)) + 1} = 3.18 \text{ 或 } 4 \text{ 跳}$$

$$\text{radius} = \frac{1.59\log(129)}{\log((129)(0.1875)(0.88)) + 0.5} = 2.77 \text{ 或 } 3 \text{ 跳}$$

最长的和最短的弱联系相差1跳。在Pointville中，人们最多通过4个中间人就可以认识任何一个其他人。Pointville是一个紧密地结合到一起的一组人。更重要的是，Pointville的直径相对较小，即使当每个人认识平均 $\lambda = 6$ 个人而非24个人。按照这里设计的近似，这种不友好的Pointville版本的直径为6跳，中心人员的半径为5跳。谣言在Pointville中很可能会很快地传播。Flatland的平均紧度可以通过相同的方法使用近似方程估计：

$$100(\text{closeness}(\text{Pointville})) = 0.2(1 - \text{density})\lambda' + 22$$

$$r = \frac{1.275\log_2(n)}{\log_2(\lambda) + 1.275} = 1.525$$

代入到紧度的方程后就得到：

$$100(\text{closeness}) = 0.2(1 - 0.1875)\,24^{1.525} + 22 = (0.2)(0.8125)(127.5) + 22 = 42.7$$

对使用程序Network.jar的5次尝试结果进行平均就得到平均紧度值为40.1。对于密度小于50%的情况，近似方程能很好地适用，但是对于更大的密度就不准确了。

像Pointville这样的团体，随着每个人认识的人的平均数的提高变得更加紧密。Stanley Milgram（Milgram，1967）进行的“六度分隔”实验基于数学事实。将少量随机性引入到假想的社会网络后产生了惊人的结果。即使网络是稀疏的，节点之间的“距离”也会急剧下降。进一步来讲，我们可以从图4-7中看出，随机Pointville紧度峰值在0～100范围中的40左右，在接近25%的密度处。随着社会联系的增加，中介的大小和数量快速降低——因为平均路径长度快速降低。

4.6 规则网络的随机性

Pointville的社会网络分析显示相对少量的认识就会导致相对较高的中心性和短的弱联系。当将少量的随机链路添加到规则网络时，网络中心性会发生什么变化？规则网络的直径、半径和平均路径长度会像随机网络一样急剧下降吗？

对于 $n = m = 100$ 的环形网络，考虑每次向环上添加一条随机链路时，对直径、半径和平均路径长度的影响。每条随机链路增加熵并降低直径、半径和平均路径长度，如图4-9所示。随着随机性提高（由熵的线性增加所指示），直径、半径和平均路径长度快速减少。实际上，大多数降低是在添加前4条或5条随机链路时发生的。图4-9演示了少量随机性对规则网络的巨大影响。小世界效应主要是由于将少量的随机性插入到了规则网络中所引起的。该效应是纯数学的，而与人性、地理、物理或生物属性无关！

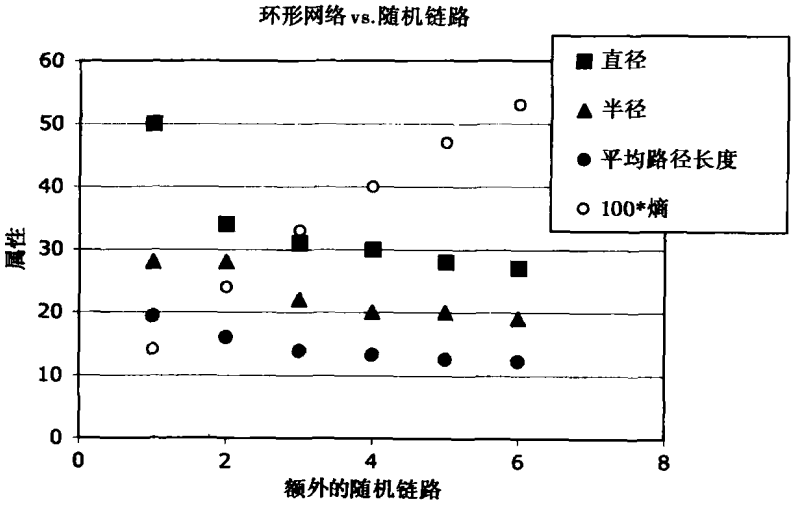


图4-9 随着随机链路数的增加，对直径、半径、平均路径长度和100 × 环形网络的熵的影响

4.7 分析

很少自然现象是纯随机的。那么，为什么还要研究随机性？答案在于：为了建立可以与非随机现象相比较的一种基线。大多数实际系统包含结构——它们很少是纯的随机系统。例如，电路的布线图、道路和铁路、水利系统以及其他大多数复杂系统，它们都是非随机网络。将纯的随机系统与结构化系统比较就能揭示结构化系统的本质。

随机网络区别于非随机网络的特性有哪些？在本章中，我们已经分析了“随机性”区别于结构化的5种主要特性：度序列、熵、平均路径长度、紧度和聚类系数。结构化网络与随机网络在这些属性上有何区别？图4-10比较了环形网络与随机网络，分别沿着4个属性轴：熵、平均路径长度、hub度和聚类系数。紧度的比较将要等到我们研究过无标度网络和小世界网络后再进行，但是它也将区别于其他类网络。

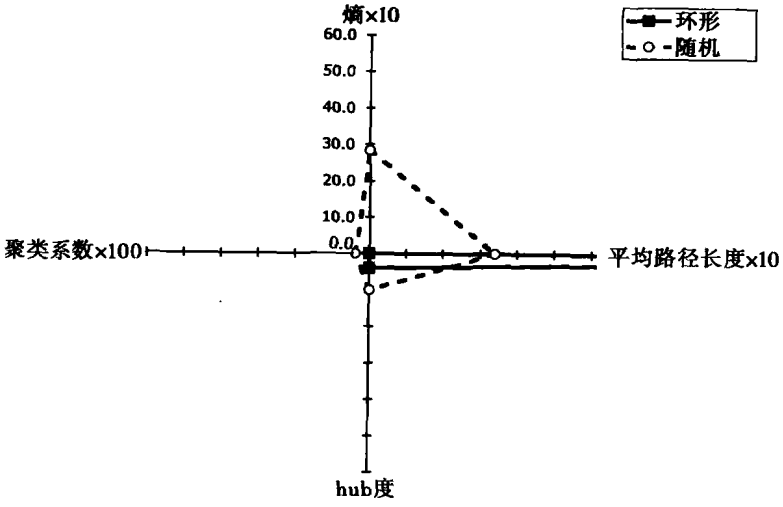


图4-10 比较环形网络与随机网络的Kiviat（基维亚特）图。环形网络（实线），$n = 100$，$m = 100$。随机网络（点画线），$n = 100$，$m = 200$，将三次随机生成的网络进行平均

以 Kiviat 图（在微软 Excel 中又称雷达图）画出的数据显示了 $n = 100$、$m = 100$ 的环形网络与 $n = 100$、$m = 200$ 的随机网络之间的主要不同。这种数据是在每种情况下实验三次后平均得到的。

Kiviat 图用来沿着 k 轴画出 k 属性。在这种情况下，$k = 4$：分别为熵、平均路径长度和 hub 度、聚类系数每个分配一个轴。熵、平均路径长度和聚类系数分别放大 10、10 和 100 倍，以便使得测量沿着相似的比例具有可比较性。

图 4-10 清楚地显示了随机网络和结构化（环形）网络之间的区别。详细地讲，随机网络具有高熵值、小的平均路径长度值和低的 hub 度值。相反，结构化环形网络具有零或接近于零的熵、聚类系数和 hub 度，但是具有相对高的平均路径长度——在图 4-10 中显示放大。详细地讲，环形网络的平均路径长度为 25 跳，随机网络的平均路径长度大约为 3.3 跳。

或许，随机网络类中最突出的属性就是典型的高熵 I。但是我们在本章中学过，随机网络的熵依赖于网络的密度。随机网络可以显示出低熵以及高熵。只有当密度接近 50% 时，随机网络中的熵才会最大化。因此，没有网络密度的知识，概括总结就会产生误导。

假定一个合理密度的随机网络落在 Kiviat 图中的第一象限——图 4-10 中的右上部分。结构化网络位于沿着平均路径长度轴，如图 4-10 所示。以后，当我们研究无标度网络和小世界网络时，就会清晰地显示出：无标度网络的 Kiviat 图位于有点中心对称于原 Kiviat 图，典型的小世界网络图形成了纵向沿 Kiviat 图的水平轴排列的扁平的椭圆。在接下来的章中将研究这些形状，但是现在我们可以总结本书中研究的网络的属性如下：

结构化的：平面，沿着平均路径长度轴。

随机的：第一象限 Kiviat 图。

无标度的：圆，以原点为中心。

小世界的：拉长的或扁平的椭圆，具有比熵和 hub 度更高的聚类系数和平均路径长度。

练习

4.1 $n = 50$ 个节点和 $m = 200$ 条链路的随机网络的平均路径长度是多少？

4.2 $n = 50$ 和密度 $= 50\%$ 的随机网络的熵是多少？将你的答案与图 4-3 给出的近似进行对比。

4.3 为什么图 4-3 中在密度 50% 处（接近）对称？

4.4 建议对 ER、Gilbert 或锚定随机网络的生成过程进行“修改”，以便保证由这些算法生成的每一个网络是强连通的。

4.5 当 $n = 50$，平均节点度 $\lambda = 5$ 时的随机网络的直径是多少？

4.6 保证 $n = 100$ 个节点的随机网络的直径不超过 5 跳的最小链路数等于多少？

4.7 对于 $n = 100$ 个节点的网络，每个节点平均度 λ 为多少时才能保证网络的直径为 5 跳？

4.8 对于 $n = 100$，$m = 500$ 的随机网络，其链路效率是多少？

4.9 对于 $n = 1\,000$，$m = 10\,000$ 的随机网络，其聚类系数是多少？

4.10 需要多少条链路才能将 $n = 10\,000$ 个节点的随机网络的聚类系数提高到 0.33？

4.11 对于 $n = 200$，$m = 500$ 的随机网络，其平均紧度是多少？

4.12 对于 n 个节点的随机网络，需要多少条链路才能保证平均紧度为 30？

小世界网络

小世界网络是具有高聚类系数、相对较短的平均路径长度和可以扩展的熵的稀疏网络。小世界网络的熵，可以降低到接近结构频谱的结构化尾部，也可以提高到更加接近随机网络。如此一来，小世界网络处于规则网络和随机网络之间。这种属性让数学家着迷，并激发出有关物理学、生物学和社会网络中小世界网络拓扑和行为之间联系的许多猜测。

在这里首先对由 Stanley Milgram 所观察到的小世界效应详细加以研究。简而言之，小世界效应就是随着少数随机链路添加到有结构的网络上，平均路径长度会快速减少。深入研究一下会发现，通过增加少数随机链路会使路径长度极大地减少。直观来讲，随机链路倾向于对分，有效地将相对的两个 50% 网络之间的距离对半分。添加第二条随机链路起到同样的效果——有效地降低距离 25%，等等。因此，小世界效应的六度分隔可能最好称为“50% 消除隔离”。

小世界网络的基础是由这里描述和研究的 Watts-Strogatz（WS）过程生成。WS 算法的典型形式是从 2 - 规则网络开始的。小世界网络是从 2 - 规则网络通过随机地重联 pm 条链路涌现的，这里 p 就是重联概率。随机重联将（有限的）随机因素注入到规则网络中。结果证明，非常少量的随机性会影响很大。典型地，小世界是从纯规则向有点随机 $p^* = 1\% \sim 4\%$ 转换而来。这种相变阈值或转换点在不同的应用中具有不同的意义。例如，相变声称能解释 Ising 效应——有色金属材料中的磁极性的转变。它也被用来解释为什么小世界网络要比纯随机的或纯结构化的网络更倾向于同步。

小世界网络还具有很多其他惊人的属性。例如，Strogatz 认为某种蟋蟀同步鸣叫就是一系列小世界效应（Strogatz, 2003）。因为小世界网络共享某些随机网络的属性——例如短路径长度，于是一些研究人员就急于得出结论：它们的行为也像随机网络。某些断言已经得到实验验证，而某些仍然还是虚构的东西。在我们整理有关到底哪些断言有事实根据之前，我们必须加深对小世界网络的理解。

在本章中我们将学习以下内容：

1. 随机化一个 2 - 规则网络的链路会创建基本的小世界网络。这一历史性的重要生成过程最先由 Watts 和 Strogatz 提出，并称为 Watts-Strogatz（WS）算法。这种网络被命名为 WS - 生成网络，或简称为 WS 小世界网络。

2. WS 生成的小世界的度序列分布要比同等的随机网络更高、更细：

$$h(d) = \sum_{i=1}^{\min\{d-k,k\}} B(k,i,1-p)P(pk,d-k-i)\ ;\ d \geqslant k$$

这里 k 就是底层的 k - 规则网络的大小，p 就是重联概率，$B(k,i,1-p) = C\binom{i}{k}(1-p)^i p^{k-i}$，$P(pk,d-k-i) = (pk)^{d-k-i} \times \exp\left(\frac{-pk}{d-k-i}\right)!$，$C\binom{i}{k}$ 是组合函数。随着重联概率 p 的增加，分布变得更短、更宽。

3. 小世界网络的熵可以从接近结构化（低熵）扩展到全随机网络（高熵）。熵可以由设计

者通过选择 WS 重联概率 p 或链路数（密度）来加以调整。对于最初的 2 - 规则起始网络，2 - 规则 WS 小世界熵近似于

$$I_{WS}(p) = A\log_2(100p) = O(\log(p))$$

这里 A 为依赖于网络规模 n 的系数。熵也是密度 $= 2k/n$ 的函数。对于一般的 k - 规则 WS 小世界和固定的重联概率，我们得到：

$$I_{WS(\text{density})} = A\log_2(B(\text{density})) - C = O(\log(\text{density}))$$

这里 A、B、C 为依赖于网络规模 n 的系数。

4. 随着重联而导致熵的增加，路径长度快速减少。2 - 规则 WS 小世界的平均路径长度通过 Newman-Moore-Watts 方程加以近似：

$$\text{avg\_path\_length(SW)} = n\frac{fr}{2k} = \left(\frac{2n}{\beta k}\right)\tanh^{-1}\left(\frac{r}{\beta}\right)$$

这里 $r = 2pm$，$\beta = \text{sqrt}(r^2 + 4r)$。对于小的 p 我们将导出更简单和更精确的幂律近似：

$$\text{avg\_path\_length(r)} = n/(4k/r^q)$$

这里 $r = pm = pkn$，$q =$ 常数。对于 $k = 2$，我们通过曲线拟合得到 $q = \frac{1}{3}$。根据 $k = 2$ 的仿真数据，平均路径长度与重联概率的立方根成反比：

$$\text{avg\_path\_length} = O\left(\frac{1}{\sqrt[3]{p}}\right)$$

5. 小世界网络聚类随着熵的增加快速减少。对于 k - 规则小世界，Newman 和 Watts 建议：

$$CC(k\text{ - 规则}, p) = 3k\frac{k-1}{2k(2k-1) + 8pk^2 + 4p^2k^2}$$

当熵为零，重联概率 $p = 0$，则聚类系数 $CC(k\text{ - 规则}, 0) = 3((k-1)/2(2k-1))$。但是，Barrat 和 Weigt 推导出了更加简单的方程，以较少的计算给出了精确的近似（Barrat，1999，2000）：

$$CC(k,p) = CC(k\text{ - 规则}, 0)(1-p)^3$$

我们说明聚类系数不仅是所有小世界网络的一个内在属性，而且还是底层 k - 规则网络的属性。在 WS 生成过程中，如果使用超立方或超环形网络替代 k - 规则网络，那么聚类就会受到很大影响。因此，我们得出结论，小世界效应主要是由于当规则网络的少数链路随机化时发生的对分。

6. 小世界网络的平均紧度增加、减少，然后再次上升，按照密度 d、规模 n、平均度 λ 以及重联概率 p：

$$\text{closeness(small world)} = O(\text{density}(z))$$

这里 $z = \lambda^r$，$r = O\left(\frac{\log(n - (1-p))\lambda}{\log(\lambda)}\right)$。

这就意味着小世界网络中的紧度与随机网络不同，既与密度又与底层初始 k - 规则网络的规则性相关。与直觉相反，节点的紧度没有提高信息快速找到通过小世界网络的路径的能力。

7. 小世界网络展示了小世界效应，因为维持底层规则网络的聚类系数的同时，平均路径长度突然减少。由于随机的二分，2 - 规则 WS 小世界网络从规则向随机拓扑迁移的转换重联概率接近 $p^* \sim O(1/n)$。这就标记着从“最结构化的”向“最随机的”拓扑迁移。正因为如此，小世界网络成了相迁移现象的最好模型，例如液体变为固体，或惰性铁变成磁铁。

8. 使用本地节点属性的网络导航采用最大度选择算法，要比采用最小半径、最大紧度或随机选择算法更快。在本章的学习中，WS 小世界网络显示出要比随机或无标度网络更加难以导航。同时小世界倾向于具有短的平均路径长度，它就要比同等的随机网络更难于使用本地信息

（如节点度、紧度和节点半径）找到通过小世界的最短路径。这对于分组交换通信网络来讲意义深刻。

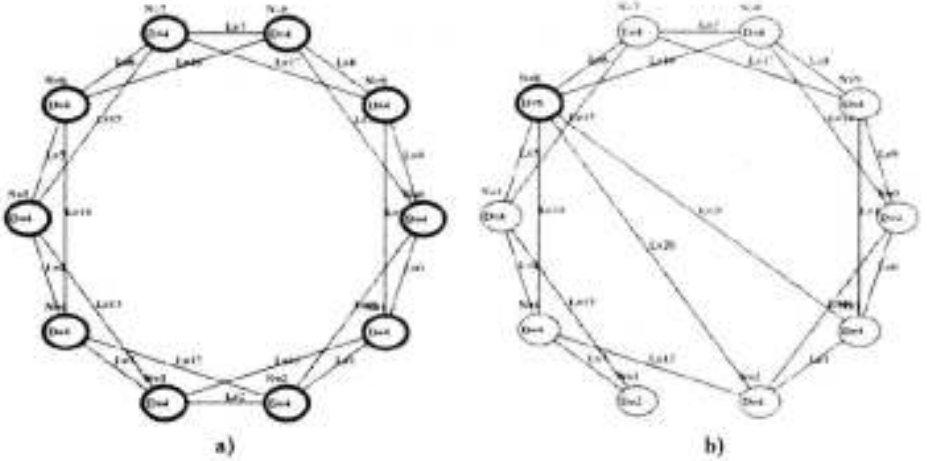
本章使用程序 Network. jar 产生并分析这里描述的每一个例子。本章中给出的用以实现生成过程的 Java 方法是 Network. jar 中找到的代码的简化版本。

5.1 生成一个小世界网络

小世界网络是可扩展的（它们几乎像规则网络一样结构化，从而造成很低的熵），或者是某种程度的随机性（具有可以与随机网络相比较的高熵）。熵的扩展性是由重联概率 p（也就是用以确定将每条链路的一端交换连接到一个随机选定的节点上的概率）确定的。重联概率与用来生成 Gilbert 随机网络的链路概率相似，但是却不是用来确定是否将一条链路包括到网络中，重联概率 p 是用来确定是否将链路的一端交换到随机选定的节点的。

在本节中，通过以概率 p 将 k －规则网络的链路重联，我们提供一组简单的用来将 k －规则图转换成小世界网络的微规则。最初 k －规则网络的所有节点都等于 $\lambda = 2k$。因此度序列分布在 $d = 2k$ 处产生尖峰，但是该尖峰随着链路的重联变得扁平并扩展开来。每条链路以概率 p 被访问并重联（连接到随机选择的头节点上）。最后，就留下 $(1-p)m$ 条链路（部分最初的 k －规则网络），并随机化剩下的 pm 条链路。结果产生的小世界部分是结构化的、部分是随机的——因此它的度序列分布落在 k －规则网络峰值和随机网络的泊松分布之间。

WS 小世界网络可以用少量随机性（小 p）或大量随机性（大 p）生成，具体根据 p 值而定，如图 5-1 所示。最初，网络与图 5-1a 相似。重联过 pm 条链路后，网络与图 5-1b 相似。通过重联 20 条链路中的 10% 而导入了少量的熵。两条重联链路的每个头部转向到随机选择的节点上。



a) b)

图 5-1 WS 小世界生成从 2－规则网络（图 5-1a）开始然后演变成半随机和半结构化的“轻微的小世界”网络（图 5-1b）；$n = 10$，$m = 20$

令人惊讶的是，两条链路的重定向对几个属性有很大的影响：度序列分布、直径、半径和平均路径长度。对于较小的重联概率 p，大多数网络仍保持结构化的。因为 k －规则网络具有高的聚类系数，结果小世界网络也具有高的聚类系数。但是即使少量的重联也会很大地减少网络直径、半径和平均路径长度，同时保留大多数的 k －规则网络内在的聚类，这称为小世界效应。

5.1.1 Watts-Strogatz（WS）过程

Watts 和 Strogatz 于 1998 年设计了 Watts-Strogatz 小世界网络生成过程，故此得名 Watts-

Strogatz 或 WS 小世界网络。Newman 等人修改了过程以避免孤立的节点和组件，从而得出了 NSWS 模型（Newman, 2000a, 2000b）。Newman 等人的生成过程与 Gilbert 随机网络生成过程相似，因为它是用了相似的“锚定”算法以避免将网络隔离成组件。WS 生成过程假定一个初始的 2 - 规则网络，但是我们对过程加以推广以允许密度变化，这是通过设置 $k = (n/m)$ 并从 k - 规则网络（$\lambda = 2k$）开始来实现的。

WS 生成过程很简单——从一个 2 - 规则网络开始并重联一定百分比的链路加以“随机化”。最初的 2 - 规则网络具有格子结构，如图 5-1a 所示，从而导致小世界网络具有某些规则和某些随机结构，如图 5-1b 所示。重联就是用随机选定的不同节点替代链路的头节点。访问每条链路并以概率 p 重联，因此小世界网络演变成一种半结构化、半随机的网络。平均来讲，pm 条链路是“随机”的，$(1-p)m$ 条链路是结构化的。

WS 生成过程

1. 给定 n、重联概率 p 和 $k = 2$，通过将 n 个节点中的每一个连接到直接邻居和邻居的邻居生成一个 k - 规则图。该网络有 $m = 2n$ 条链路（$\lambda = 4$，密度 = $(4/n)$）。

2. 对于每一条链路，$\mu = 1, 2, \cdots, m$，以概率 p 重联链路 μ 如下。如果（Math. random() $< p$），断开头 head(μ) 并将它重联到不同的随机选择的节点上。避免（$v_{\text{random}} = \text{head}(\mu)$）和重复链路。否则，什么都不做。

这种生成过程简单，但是不能保证是连通的网络。如果连接到一个节点的所有链路重联，那么该节点就很可能被孤立了。因此，可以将很多增强版本（各种研究人员所建议的）应用到基本算法上，以便去掉这种不足。例如我们可以修改 WS 过程以避免当要从一个节点上删除最后一条链路时的重联。这与前一章中描述的将锚定的随机网络修改应用到 Erdos - Renyi 过程一样。这种增强的算法留给读者作为练习[⊖]。

以下 Java 方法实现 WS 网络生成过程。最初的 2 - 规则网络是由方法 NW_doCreateNodes()、NW_doRing(1, true)和 NW_doRing(2, true)创建的。方法 NW_doRing()被调用了两次：一次是在将链路插入到网络以便形成一个 1 - 规则结构，再次则是构成 2 - 规则结构时。这样就建立了如图 5-1 所示的初始 2 - 规则网络。

生成过程的步骤 2 是取自程序 Network. jar 的 NW_doCreateSmallWorld()中余下的代码实现的。每条链路访问一次，以 p = nConnectionProbability（这是一个由用户设置的全局常量）概率重联。余下的代码避免了循环和重复链路。

```
public void NW_doCreateSmallWorld(){
   NW_doCreateNodes(nInputNodes);                    //Create 2-regular network
   NW_doRing(1, true);
   NW_doRing(2, true);
   for(int j = 0; j < nLinks; j++){                  //Rewire ring links
    if((int)(100*Math.random()) < = nConnectionProbability){
       boolean done = false;
       while(!done){
        int to_node = Link[j].head;
        int from_node = Link[j].tail;
        int new_node = (int)(nNodes * Math.random()); //Random head
          while(new_node == to_node)
            new_node = (int)(nNodes * Math.random()); //New head
```

⊖ 因其简洁性和固定的密度属性，WS 模型被用于全书。

```
            if(NW_doAddLink(node[ from_node] .name,
            node [ new_node] .name)){
               NW_doCutLink(j);                    //Erase old link
                        done = true;
          }
        }
      }
    }
}//NW_doCreateSmallWorld
```

在该方法中的 while（！done）循环似乎是不必要的，但是当方法 NW_doAddLink()发现了一条重复链路时它保证了重联。回顾方法 NW_doAddLink()检查重复链路并在方法 NW_doCreateSmallWorld()试图添加一条重复的链路时返回 false。重复 while 循环直到一条非重复的、非循环的链路成功地重联。（注意：如果 2－规则的起始网络是 n －规则网络或完全网络，就会导致一个无限循环。如果后跟 WS 规则，这就不会发生。）

最后要注意的是，重联是通过插入一条新的链路并删除老的链路实现的。因为重复的原因，很有可能新的链路不能插入。因此在成功地插入前不会删除老的链路。如果所有链路在选择后成功地重定向，则重联前后网络密度保持不变：$d(4/n)$ 。

5.1.2 一般的 WS 过程

小世界密度可以通过修改 WS 的初始化过程加以调整。我们不再从 2－规则网络开始，而是从 k －规则网络起始，这里 $k = m/n$ 。程序 Network. jar 提供了第二种构建小世界网络的方法，除了将链路插入到 k －规则起始网络的循环算法之外，这两种方法非常相似。k 值根据节点数和链路数由用户指定。这就要求插入 $k = m/n$ 个环生成最初的 k －规则网络。

方法 NW_doCreateGSmallWorld()插入环以便构成 1－规则、2－规则、3－规则……网络，直到 nInputLinks 条链路插入到 k －规则网络为止。当随机选择的链路由于重复不能重联时，该实现版本也尝试找到一条新的链路重联。这种修改通过使用可变的 switched 发信号表示是否尝试重复重联来加以实现。

```
public void NW_doCreateGSmallWorld(){
   NW_doCreateNodes(nInputNodes);
   int offset = 1;                              //Ring number
   while(offset > 0 && offset < nInputNodes/2){
     for(int i=0; i < nNodes; i++){    //Ring around the Rosie
       if(nLinks < = nInputLinks){offset = 0; break;}
       int k = i+offset;
       if(k < = nNodes) k- = nNodes;
       NW_doAddLink(node[i].name, node[k].name); //Ignore duplicates
     }
     if(offset > 0) offset++;
   }
   for(int j = 0; j < nLinks; j++){       //Rewire && eliminate infinite loop
     if((int)(100*Math.random()) > = nConnectionProbability){
        int to_node = Link [ j] .head;
        int from_node = Link [ j] .tail;
        int new_node = (int)(nNodes * Math.random());  //Pick new node
        while(new_node == to_node)
            new_node=(int)(nNodes * Math.random());
        boolean switched  =  false;
```

```
            while(!switched){
            if(NW_doAddLink(node [ from_node] .name, node [ new_node] .name)){
               NW_doCutLink(j);                              //Erase old link
               switched = true;
            }
            else new_node = (int)(nNodes * Math.random());
            }
          }
       }
    }//NW_doCreateGSmallWorld
```

方法 NW_doCreateSmallWorld( )从 2 - 规则起始网络开始生成小世界网络，因此我们称之为 2 - 规则 WS 网络。相似地，方法 NW_doCreateGSmallWorld()从任意一个 k - 规则起始网络生成一个小世界，因此我们称之为 k - 规则 WS 网络。我们知道 $k = m/n$。因此给定 nInputNodes = n 及 nInputLinks = m，方法 NW_doCreateGSmallWorld()就可以产生一个任意 k 值的 k - 规则 WS 网络。k - 规则 WS 过程用来研究小世界属性的变化，它是重联概率 p 和密度 d 的函数。

5.1.3 小世界网络的度序列

小世界是部分 k - 规则和部分随机网络的混合。因此小世界网络的拓扑介于 k - 规则网络和随机网络之间。在这种情况下，直觉上的小世界网络的度序列分布要比同等的随机网络更高更细是正确的[⊖]。图 5-2 演示了小世界网络和同等随机网络的这种事实。小世界的度序列分布随着重联概率 p 的增加变得更平更广，直到两种分布匹配为止。

Barrat 和 Weigt（Barrat, 2000）通过以下观察导出了一种小世界度序列分布的紧凑形式的表达式：

1. 如果典型节点 u 的链路没有重联，它的度就不会变化，如果其他节点的链路重定向到 u 时度就会增加。

2. 不重联链路的概率服从二项式分布 $B(k,i,(1-p))$，这里 $k = m/n$，i = 不重联的链路数，p 为重联概率。

3. 其他节点的链路重联到节点 u 的概率服从泊松分布 $P(\lambda_1, d-k-i)$（$d \geqslant k$），这里 $\lambda_1 = pk$ 是重定向链路的期望值，d = 度数，i = 重联到 u 的链路数。

4. 节点 u 度数增加的概率等于联合概率 $B(k,i,(1-p))P(\lambda,d-k-i)$。

5. 度分布 $h(d)$ 等于在 $i = 1,2,3,\cdots,\min\{d-k,k\}$ 条链路上的联合概率的总和。

重联后具有 d 条链路的节点分布就是总和：

$$h(d) = \sum_{i=1}^{\min\{d-k,k\}} B(k,i,(1-p))P(\lambda_1,d-k-i);d \geqslant k$$

这里 $\min\{d-k,k\}$ 等于 $d-k$ 和 k 中最小的：

$$\lambda_1 = pk$$

$$B(k,\ i,\ (1-p)) = C\binom{i}{k}(1-p)^i p^{k-i}$$

$$P(\lambda_1,d-k-i) = (pk)^{d-k-i}\exp\frac{-pk}{(d-k-i)!}$$

$$C\binom{i}{k} = \text{组合函数} = \frac{k!}{i!(k-i)!}$$

⊖ 同等的意味着“同一密度”。

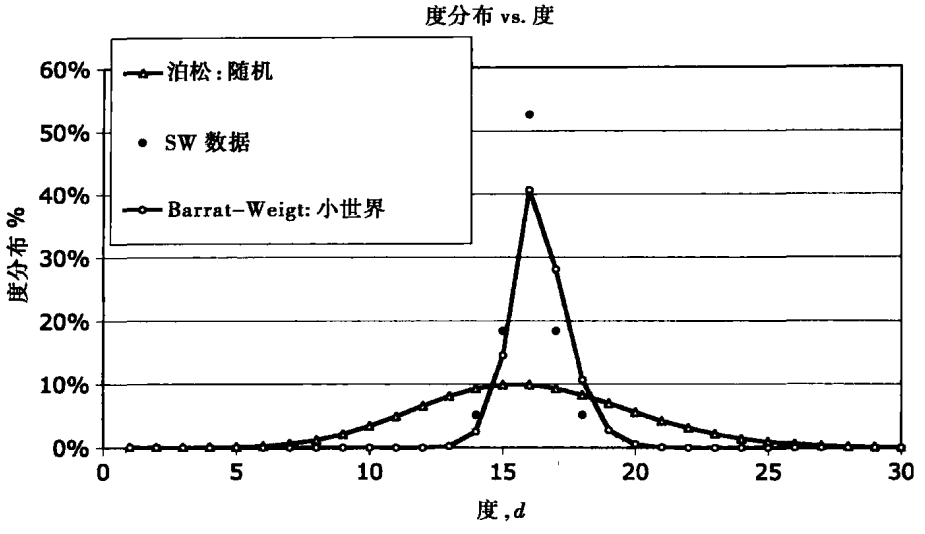


图 5-2 对于 $n = 50$、$m = 400$ 和重联概率 $p = 5\%$ 时的随机网络和小世界网络的度序列分布

该表达式不是为 $d < k$ 定义的，因此对于所有 $d < k$ 的及所有没有落在[k , min{ $d-k$, k }]内的 i 值，使 $h(d) = 0$。例如，对于图 5-2 的小世界网络来讲 $h(15)$ 等于多少？

$$d = 15$$

$$p = 0.05$$

$$k = \frac{m}{n} = \frac{400}{50} = 8$$

$$\min\{d-k,k\} = \min\{15-8,8\} = 7$$

$$\lambda_1 = pk = 0.05(8) = 0.4$$

$$h(15) = \sum_{i=1}^{7} \{B(8,i,(0.95))P(0.4,15-8-i)\}$$

$$B(8,i,0.95) = C\binom{i}{k}(0.95)^i(0.05)^{8-i}$$

$$P(0.4,15-8-i) = (0.4)^{15-8-i}exp\frac{-0.4}{(15-8-i)!}$$

$$h(15) = \sum_{i=1}^{7}\left\{C\binom{i}{k}(0.95)^i(0.05)^{8-i}(0.4)^{7-i}\exp\frac{-0.4}{(7-i)!}\right\}$$

表 5-1 中的表单计算显示了图 5-2 中的网络和 $k = 8$ 时是如何获得 $h(15) = 14.5\%$ 的。实验值 $h(15) = 18.4\%$ 是平均 5 个由程序 Network. jar 产生的小世界网络而来的。一般来讲，对于本书研究的网络规模，Barrat-Weigt 方程要比观测到的值稍小——将在图 5-2 中画出的数据点与从 Barrat-Weigt 方程中获得的曲线相比较。

表 5-1 $h(d)$ 项目总结

d	min $(d-k, k)$	$h(d)$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$
10	2	0.0%	0%	0%	0	0	0	0	0	0
11	3	0.0%	0%	0%	0%	0	0	0	0	0
12	4	0.0%	0%	0%	0%	0.02%	0	0	0	0
13	5	0.3%	0%	0%	0%	0.01%	0.24%	0	0	0
14	6	2.5%	0%	0%	0%	0.01%	0.19%	2.31%	0	0

（续）

d	min $(d-k, k)$	$h(d)$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$
15	7	14.5%	0%	0%	0%	0.00%	0.08%	1.85%	12.55%	0
16	8	40.6%	0%	0%	0%	0.00%	0.02%	0.74%	10.04%	29.81%
17	8	28.1%	0%	0%	0%	0.00%	0.00%	0.20%	4.02%	23.85%
18	8	10.7%	0%	0%	0%	0.00%	0.00%	0.04%	1.07%	9.54%
19	8	2.8%	0%	0%	0%	0.00%	0.00%	0.01%	0.21%	2.54%
20	8	0.5%	0%	0%	0%	0.00%	0.00%	0.00%	0.03%	0.51%

5.2 小世界网络属性

小世界具有相对小的平均路径长度、高的聚类系数和可以调整的熵（根据重联概率 p 可以更改）。少数链路的“随机化”效应显示在图5-1中之前和之后的网络中。相对较小的随机性就会对路径长度、聚类和熵有很大影响，这是小世界网络最显著的属性。

图5-1a网络是完全结构化的，因此它的熵等于零。除了两条链路从其节点拆除一端并将它连接到随机选择的新节点上，按照上述生成过程随机地重联之外，图5-1b中的网络相同。在这种情况下，$p = 10\%$，$m = 20$，因此两条链路重联。少量重联将熵从零更改为接近2.12比特。

直观上讲，图5-1a中规则网络结构上的小更改应该对整个网络有较小影响。但是这种直觉是错误的！相对较小的更改有着很大的影响，如表5-2中所见。2－规则网络具有零熵，并且图5-1b所示的“轻微的小世界”网络比起它的2－规则网络具有更小的平均路径长度，但是具有很高的熵。一般来讲，轻微的小世界网络要比它之下的规则网络具有高的聚类系数和更低的平均路径长度。

例如，表2-2和表2-3（第2章）比较了小世界网络和结构化网络的聚类系数和平均路径长度。表2-2包含了比同等随机网络聚类高100～1000倍的实际小世界网络的例子。表2-3中小世界网络的平均路径长度是同等2－规则网络的三分之一。实际上，随机网络和无标度网络显示了同样的趋势。

这些观察展示了一个主要的概念：引入少量随机性或熵到结构化网络中，会导致结构化网络属性的巨大变化。这种效应——称为*小世界效应*——对于网络科学来说非常重要，它建模（或解释）了在随后章节中详细探索的很多现象。

表5-2 通过重联2－规则网络产生的WS小世界网络与超环形网络的对比

属性	2－规则	WS小世界	超环形	超环形→SW	随机
平均路径长度	3.5	2.87	2.5	2.37	2.33
聚类系数	0.500	0.363	0	0.055	0.169
熵	0	2.83	0	3.61	5.82

网络属性随着熵的提高而快速更改带来了很多有关WS小世界的问题，以及从显著的结构化网络到部分结构化然后是完全随机网络的迁移。例如，这种现象也发生在从其他结构化网络到小世界的迁移中，那么它依赖于初始的2－规则格子吗？如果WS生成过程是从超环形网络开始又如何？这种研究问题部分地由以下实验解决。从一个5×5超环形网络开始，并应用与WS生成过程相同的重联微规则，导致的网络属性是什么？

表5-2总结了 $n = 25$、$p = 10\%$ 的结果。“2－规则”列包含了 $m = 50$ 的规则网络的属性值。“WS小世界”列显示了 $n = 25$、$m = 50$、$p = 10\%$（5条链路重联）的结果。“超环形”列包含 $n = 25$、$m = 50$ 的规则5×5超环形网络，“超环形→SW”列包含从超环形网络而非2－规则网

络开始的实验生成过程属性值，然后再迁移到 WS 小世界网络。最后一列包含 $n = 25$ 、$m = 50$ 随机网络的结果。我们使用随机网络作为基础进行比较。

表 5-2 中的结果显示，最初的配置对熵和平均路径长度有影响，但是在由 WS 微规则生成的网络中，从超环形网络开始要比从 2 - 规则网络开始的网络聚类系数低得多。标准 WS 生成过程开始和结束都有很高的聚类。“超环形 → SW” 生成过程开始和结束都有较小的聚类。换句话来讲，这与起始网络有关。WS 网络生成算法产生了聚类系数由 2 - 规则网络拓扑决定的小世界。总的来讲，底层的 k - 规则网络决定了导致的小世界的聚类系数。

在表 5-2 中，2 - 规则网络和 WS 小世界网络具有最高的聚类系数。这两个网络和规则超环形网络具有最低的熵。根据这些属性，由 WS 网络过程生成的小世界网络为结构化网络。另一方面，超环形 → SW 网络和随机网络具有最高的熵和最低的平均路径长度。根据这些属性，超环形 → SW 网络和随机网络是随机的。这似乎是一个悖论！

小世界网络的确是具有额外少量随机性的结构化网络。小世界网络的某些属性是从小世界底层的结构化网络继承而来的，其余是一系列重联。小世界的平均路径长度受制于导入的随机性，但是其他属性如聚类系数受制于起始网络的拓扑。

认真学习网络科学的学生，可能想要进一步探索其他可能性。从二叉树或其他根据 WS 过程重联的规则网络产生的聚类系数是多少？初步指示反映出小世界网络的最终形状极大地受最初网络的形状影响。在第 7 章中我们演示如何产生具有几乎任何等级的聚类、路径长度和熵的网络。具有这种能力后，我们就可以生成具有任意属性的“设计者网络”。

5.2.1 熵与重联概率

小世界效应对直径和平均路径长度等属性具有很大影响。因此，当注入更多的随机性时会发生什么？到达某点小世界网络会变得更加随机吗？然后随着重联概率（和熵）的增加反而会更加结构化？相似地，更改密度会修改熵和路径长度等属性。每个小世界属性与重联概率和网络密度之间有什么样的关系？

为了回答这些问题，我们进行以下实验：（1）保持 2 - 规则 WS 网络的规模和密度不变而改变重联概率 p；（2）保持规模和重联概率不变而更改密度。我们从 $k = m/n$ 的 k - 规则网络开始，并让密度 $= 2k/(n-1)$。这些仿真产生 WS 网络属性近似为重联概率或密度的函数。图 5-3 至图 5-5 显示了更改重联概率对熵、平均路径长度和聚类系数的影响，而图 5-4 显示更改密度的影响。

图 5-3 显示在 5 个 2 - 规则 WS 网络为每个重联概率 p 产生的平均熵结果（p 在半对数（semilog）x 轴上从 0 增加到 100%）。然后，我们使用直线 $y = A\log_2(100p)$ 拟合平均数据点。概率 p 是重联链路的百分比，常数 A 是确定的。实验导出来的熵与重联概率的半对数图（图 5-3）刚好拟合直线[⊖]。因为 x 轴是对数的，对于小概率 p，随机性上升非常快，然后当 p 接近 100% 时就平缓下来。接近一半熵（1.3 比特）的增加发生在 1% 和 10% 之间（在对数 x 轴上为 3.32）。这证实了先前的推断——小的随机性将会导致小世界网络或随机网络属性的很大改变。

因为熵是用比特测量的，因此我们使用以 2 为底的对数建模熵作为重联概率 p 的函数：

$$\text{Entropy}_{WS(p)} = I_{WS(p)} = A\log_2(100p); \quad p = 0.01, 0.02, \cdots, 1.0$$

对于图 5-3 的网络，$A = 0.437$。根据该公式，随机性的比特数从 0 上升到接近 2.9。将它与有 100 个符号的消息中的信息比特比较，以相同的概率 $\left(\frac{1}{100}\right)$ 发生：$I(1 \cdots 100) = \log_2(100) =$ 6.64 比特。在区间 $[1 \cdots 100]$ 中的一系列随机数包含 6.64 比特信息。$n = 100$ 的小世界包含不超

⊖ 直线就是图中显示的重联概率范围，但是对于小的 p 值，线不是直线。我们后面研究小 p 的例子。

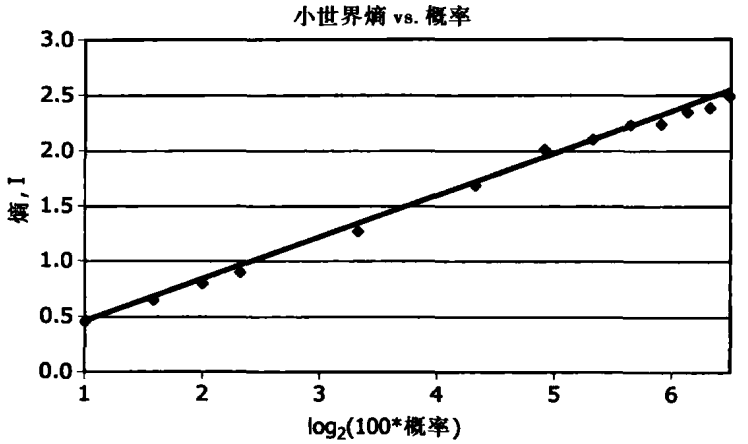


图 5-3 WS 小世界的熵与重联概率 p 的对数对比，$n = 100$，$m = 200$。重联概率从 1% 变成 100%

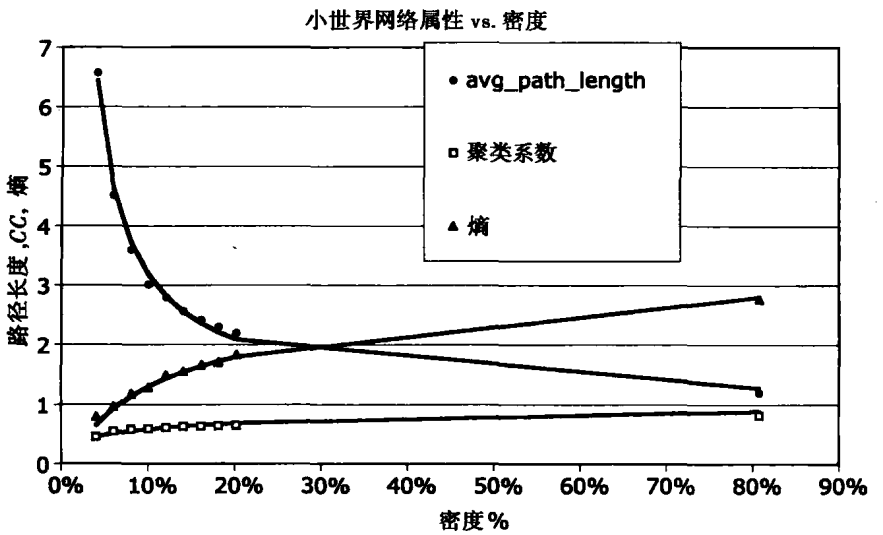


图 5-4 重联概率 $p = 4\%$，$n = 100$，$4\% \leqslant$ 密度 $\leqslant 80\%$ 的小世界网络的属性（注意：kn(密度/2)）

过 ~1/2 的同样的比特数。WS 小世界网络可以像纯随机过程那样很好地（按对数增长）扩展，但是远远低于完全随机序列数携带的熵。

当然，我们也可以通过直接将度序列分布代入到熵公式中加以计算：$I_{WS} = -\sum h(d)\log_2(h(k))$。如果我们对表 5-1 中的例子这样做，就可以得到 $I_{WS}(p = 5\%) = 2.13$ 比特。但是我们寻找一个更简单的公式，不需要像 Barrat-Weigt 公式中那样求复杂项的和。对于足够大的重联概率 p，熵随着 p 呈对数地增加。这从直观上来讲得到满足，因为据说小世界中熵比特数直接与重联概率 p 成比例：熵比特数$(p) = O(p)$。

当 $p = 100\%$ 时，WS 小世界网络与具有相同数量节点和链路的同等随机网络从熵的角度来讲是相同的[⊖]。与随机网络不同，WS 小世界的熵可以从零放大到同等随机网络的熵。从某种意义上来讲，我们可以“调整”小世界匹配我们想要的随机性。因此，熵的可扩展性是与“增加

⊖ 同等网络具有同样的密度。

随机性”相一致的。

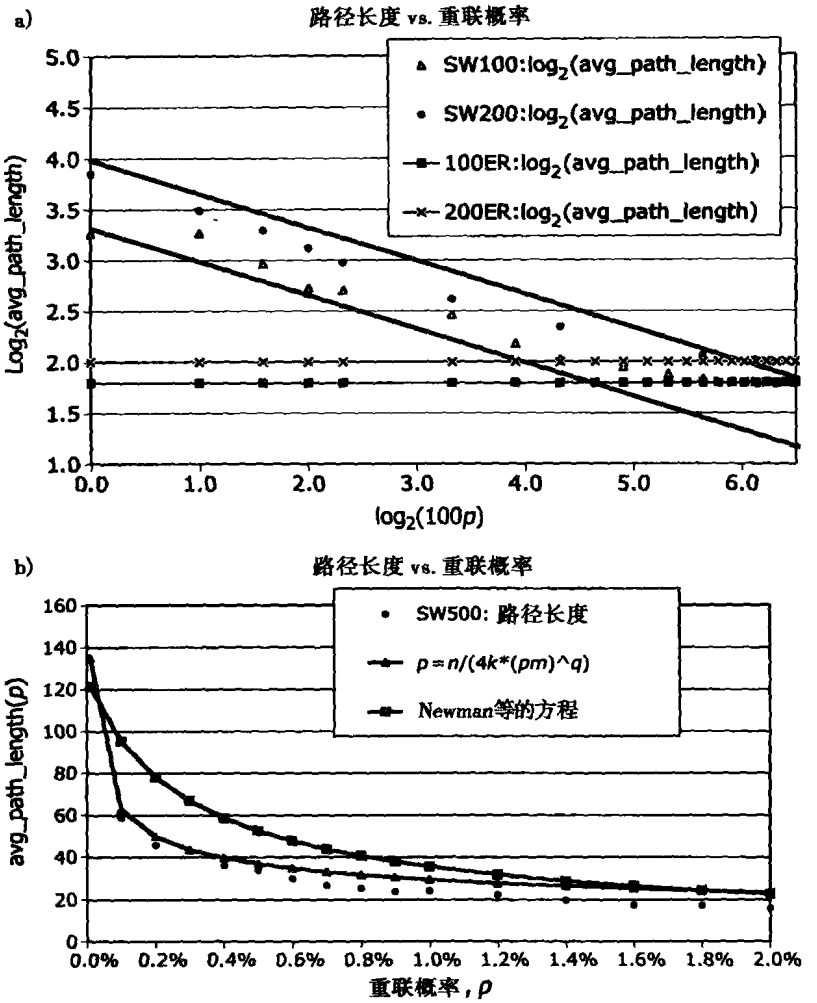


图5-5 2-规则WS网络的平均路径长度与重联概率 p 之间的关系图：a）小世界网络平均路径长度对 p（对于“较大的 p”（$n=100$、200））的log-log图，与相同密度的随机网络相比；b）小世界平均路径长度与小的 p 的2-规则WS网络，$n=500$，$m=1000$（“小 p”，不带有log-log放大）。近似方程：Newman-Moore-Watts方程，以及由作者推导出的近似

5.2.2 熵与密度

图5-4描述 k-规则WS小世界网络的熵随着底层 k-规则网络密度增加之间的关系，也就是说通过允许 $k=2,3,4\cdots$ 来更改底层的 k-规则网络，因而更改网络的密度。由于 k-规则网络的密度与 k 成比例，我们可以将 k 替换成密度，反过来就得到：

$$\text{Density}(k\text{-规则}) = 2\frac{k}{n}; k = n\frac{\text{density}}{2}; n \gg 1$$

根据对数函数 $I_{\text{WS(density)}} = A\log_2(B(\text{density})) - C$，熵随着密度的增加而非常缓慢地增加。

将从曲线拟合中获得的参数 $A=0.5$、$B=60$、$C=10$ 代入该模型并加以简化（参见图5-4），得到

$$I_{\text{WS(density)}} = 0.5\log_2(60(\text{density})) = O(\log_2(\text{density})) = O\left(\log_2\left(\text{sqrt}\left(\frac{k}{n}\right)\right)\right)$$

再次，熵比特数随着密度线性增加：

$$\text{熵比特数(denstiy)} = O(\text{density})$$

这与直觉相一致，因为它说明了小世界网络中的熵增加到同等程度，而与随机性的源无关。熵应该是相同的，与随机性的源无关。对于使用 $I(p) = 0.437\log_2(100p)$ 与 $I(\text{density}) = 0.5\log_2(60(\text{density}))$ 的小世界网络，对比重联概率和插入链路以提高密度的结果。两种近似产生可以比较的结果：

$$0.437\log_2(4) = 0.874 \text{ 和 } 0.5\log_2(2.4) = 0.632, \text{ 对于 } 4\%$$

$$0.437\log_2(80) = 2.76 \text{ 和 } 0.5\log_2(48) = 2.79, \text{ 对于 } 80\%$$

与随机网络的熵不同，小世界的熵缓慢地增加，然后随着密度或重联概率的增加而平缓下来。因此，既可以通过重联也可以通过添加链路调整小世界的熵。但是在100%处重联概率和密度之间的对数关系会很快消失，因为网络将变成完全网络。

5.2.3 小世界网络的路径长度

某些人声称小世界网络的特征或平均路径长度要比同等随机网络的小。这是真的吗？随着重联概率增加因此导致熵的增加时，小世界网络的特征路径长度会怎样变化？随着更多链路被“随机化”，小世界会变得更小吗？实际上，WS小世界网络的平均路径长度随着熵的增加呈指数减少，但是却不会减少到同等随机网络以下。这也意味着直径和路径长度缩减，因此随着更多链路被随机化小世界变得更小了。随着重联的增加，小世界变得更加随机化了，并且它的平均路径长度接近随机网络的。但是小世界网络的平均路径长度决不会小于同等随机网络的。

像熵一样，平均路径长度既受重联概率或密度的影响，又同时受两者的影响。为了证实这一点，假定一个2-规则WS网络（具有固定的密度），观察当重联概率更改时会发生什么变化。图5-4显示了通过平均特征路径长度获得的结果，从5个2-规则WS小世界网络，它们分别是以概率 p（p 像从前一样从1%到100%）重联最初的2-规则网络获得的。显然，像预期的那样，平均路径长度随着概率的增加而快速减少。但是曲线下降的形状与随机网络或无标度网络的不同，因为小世界网络是部分随机、部分规则的。

在Newman、Moore和Watts（Newman，1999a）根据下列推理导出估计以前，找出描述稍微随机的小世界网络的平均路径长度的准确方程一直是一个非常富有活力的研究课题：

1. 当不存在重联时，最初的网络是2-规则的，平均路径长度为 $n/4k$。

2. 对于非常小的重联概率 p，当 $p \geqslant (1/m)$ 后平均路径长度快速下降，如图5-4所示。

3. 在某些（早期）点 p^*，重联足够大以至于网络从大多数规则的迁移到大多数随机的。这被称为转换点 p^*，它标记网络中的相变。p^* 值对于物理学家来说有着重要意义，关联着材料的相变。

Newman、Moore和Watts（Newman，1999b；Newman，2000a）讨论了平均路径长度从最初的 $n/4k$ 值（$p = 0$）按照扩展函数 $f(r)$ 下降，这里 r 是重联链路平均数的两倍，$r = 2pm$。将 r 看做通过网络的两分捷径数的两倍：

$$f(r) = 4\frac{\tanh^{-1}(r/\beta)}{\beta}\text{；这里 } \beta = \sqrt{(r^2 + 4r)}$$

$$\text{avg\_path\_length(SW)} = n\frac{f(r)}{2k} = \frac{2n}{\beta k}\tanh^{-1}\left(\frac{r}{\beta}\right)$$

这里 $\tanh^{-1}$ 是超正切（hypertangent）函数 arc_tanh() 的反函数。对于大的稀疏网络（$n, m \gg 1$）来说，Newman-Moore-Watts方程为 $O(1/p)$。例如，假设 $n = 100$，$m = 200$，$k = 2$，密度 $= 0.04$，$p = 0.04$，如图5-4所示：

$$r = (2)(0.01)(400) = 8, \beta = \sqrt{(64 + 32)} = 9.8$$

$$\text{arc\_ tanh}\left(\frac{8}{9.8}\right) = 1.15\,,\ \frac{2n}{\beta k} = \frac{200}{2(9.8)} = 10.2$$

$$\text{avg\_path - length(SW)} = (10.2)(1.15) = 11.7 \text{ 跳}$$

将这种预测与由图5-4给出的实验值6.6相比较。Newman-Moore-Watts表达式过大地估计了平均路径长度，如图5-5b中所示，特别是当重联概率p减少时更是如此。但是，底层假设路径长度随着重联的增加而减少，意味着一种基于重联链路作为二分网络的捷径的直观方法。

最初的平均路径长度为$n/4k$，但是当重联链路时，它们就会引入将网络划分成越来越小的子图的捷径链路。详细讲来，平均一条随机链路将网络分成两部分，两条链路将网络分成四分之一大小，依此类推。但是二分是重叠的而非层次化的。因此每次重联链路，添加的额外重联链路将网络分成原来的$\frac{1}{2}$。对平均路径长度对数的影响是重联链路的对数函数。

设$r = pm = pkn$捷径链路平均以$O(\log_2(pm))$降低路径长度的对数增长，因为发生的部分二分。因此，平均路径长度的对数等于初始2-规则网络路径长度的对数，减去部分二分的效应：

$$\log_2(\text{avg\_path\_length}(r)) = \log_2\left(\frac{n}{4k}\right) - q\log_2(r)$$

这里$r = pm$，q = 常数。该表达式在log-log图上拟合成一条直线，如图5-5a所示。指数q可以近似地通过曲线拟合——要看仿真数据与从应用该近似获得的直线的匹配程度如何（对于$q = \frac{1}{3}$，$n = 100$，200的2-规则网络）。q的近似仅对$k = 2$有效。对于$k = 4$，$q = \frac{1}{6}$给出了很好的近似，导致了加倍k则半分q的推测。这留给读者作为练习。

重新调整各项就产生了幂律分布：

$$\text{avg\_path\_length}(r) = \frac{n/4k}{r^q}\text{，这里 } r = pkn$$

$$\text{avg\_path\_length(small-world},p,n) = \frac{n/4k}{(pkn)^q};\ 0 < p < 1 = \frac{n}{4k};\ p = 0$$

简单幂律近似要比Newman-Moore-Watts方程更好地拟合仿真数据，但是它假定指数q已知。对于图5-5中使用的网络规模，我们找到$q = 1/3$，因此平均路径长度与r的立方根成反比：

$$\text{avg\_path\_length}(r) = \frac{n/4k}{r^{1/3}}$$

再次使用图5-4和图5-5中的数据，$n = 100$，$m = 200$，$k = 2$，$p = 0.04$，$r = pm = (0.04)(200) = 8$：

$$\text{avg\_path\_length}(r) = \frac{\frac{100}{8}}{8^{1/3}} = \frac{12.5}{2} = 6.25 \text{ 跳}$$

Newman-Moore-Watts近似产生8跳：

$$r = 2pm = 2(0.04)(200) = 16\,,\ \beta = \sqrt{(256 + 64)} = 17.89$$

$$\text{arc\_tanh}\left(\frac{16}{17.89}\right) = 1.43\,,\ \frac{2n}{\beta k} = \frac{200}{17.89(2)} = 5.59$$

$$\text{avg\_path\_length(SW)} = 5.59(1.43) = 8 \text{ 跳}$$

实验获得的平均路径长度为6.6跳。幂律分布估计6.25跳要比Newman-Moore-Watts方程的8跳更准确。幂律分布倾向于低估路径长度，而Newman-Moore-Watts则倾向于高估路径长度。还要注意6.6跳要比同等随机网络的路径长度高得多，随机网络的平均路径长度为3.46跳。因此同

等的随机网络并非真的等同，因为熵不能被参数 p 所调整。

总之，快速减少平均路径长度及具有较大的聚类系数是小世界网络的独特特点。同等的随机网络也就是其平均路径长度小于或等于同等小世界平均路径长度的小世界网络的极端近似。小世界路径长度随着重联概率的增加而收缩，达到 $p = 100\%$ 为极限。在低重联概率值，WS 小世界路径长度与 2 - 规则网络（要比随机网络的高）相同。在高重联概率值，WS 小世界和随机网络的路径长度相同（较低）。在何处小世界幂律分布衰减，并且被随机网络路径长度对数律所接替？我们将在 5.3 节解决从大多数结构化向大多数随机网络迁移的问题。

5.2.4 小世界网络的聚类系数

在 WS 网络中通过更改密度、重联概率或者两者同时改变可以调整聚类系数。首先，我们保持密度不变，同时更改重联概率 p。当熵增加时，2 - 规则 WS 网络会发生什么变化？令人惊讶的是，聚类系数随着熵的增加而减少。另一方面，聚类系数随着网络密度的增加而增加，这是因为可以用来构成三角子图的链路数也在增加。

直觉告诉我们，小世界聚类系数应该是一个极端近似于规则网络的（高）聚类系数，另外一个极端近似于随机网络的（低）聚类系数。对于低重联概率，2 - 规则 WS 网络大多数是 2 - 规则的，而对于高概率 p，网络大多数是随机的。

最初，在重联之前，2 - 规则网络中每一节点的聚类系数是相同的。Newman、Strogatz 和 Watts（Newman，2001a）讨论初始的 2 - 规则网络的聚类系数 $CC(0)$ 等于连接到每一节点的三角数除以连通的三角数的三分之一：

$$CC(0) = 3\frac{\text{三角数}}{\text{连通的三角数}} = 3\frac{k(k-1)}{2k(2k-1)} = 3\frac{k-1}{2(2k-1)}$$

例如在最初的 2 - 规则网络中，$k = 2$，因此 $CC(0) = 3(2(4-1))^{-1} = \frac{1}{2}$。随着 k 没有边界地增加，最高可能的聚类系数是 $\frac{3}{4}$，因为 $CC(0) = (3k-3)/(4k-2) \sim (3k/4k) = \frac{3}{4}$。因此，小世界网络的聚类系数落在区间 $[0.5, 0.75]$。

当小世界网络从重联中涌现时，pm 条链路将被重联，这就意味着某些链路已经偏离了三角子图而其他保持最初的不变。这将导致随着对应重联概率 p 的增加，聚类会下降。图 5-6 证实了该实验。当 $p = 0$ 时，小世界聚类系数等于 2 - 规则网络的 (0.5)，因为在 WS 生成过程中 $k = 2$。随着 p 的增加，聚类从 $CC(2 - \text{规则}) = 0.5$ 下降到 $CC(\text{random}) = 0.02 \sim 0.04$。当 $p = 1.0$ 时，WS 小世界聚类系数等于随机网络的聚类系数。因此，我们可以通过调整 p 来调整 k - 规则 WS 网络的聚类系数。

图 5-6 是由 2 - 规则 WS 过程生成的 5 个网络平均聚类系数获得的，并为两种网络画出平均数据与重联概率图：一种 $n = 100$ 个节点，另外一种 $n = 200$ 个节点。平均度 $\lambda = 2(m/n) = 2k = 4$，通过调整链路数 $m = 200, 400$ 保持不变。

Newman 和 Watts 使用组合方法导出了一种聚类系数的复杂方程（Albert，2000）：

$$CC(k - \text{规则}) = 3\frac{k(k-1)}{2k(2k-1) + 8pk^2 + 4p^2k^2},\ p = \text{重联概率}$$

从图 5-6 中很明显地可以看出，这种近似过大地估计了 $k = 2$ 时的聚类系数。Barrat 和 Weigt 给出的更简单的近似给出了更好的简单假设，它提供了更精确的结果——不再重联的三角子图中聚类系数与概率成正比（Barrat，2000）：

$$CC(k - \text{规则}) = CC(0)(1-p)^3;\ \text{其中}\ CC(0) = 3\frac{k-1}{2k(2k-1)}, k = 2, 3, \cdots$$

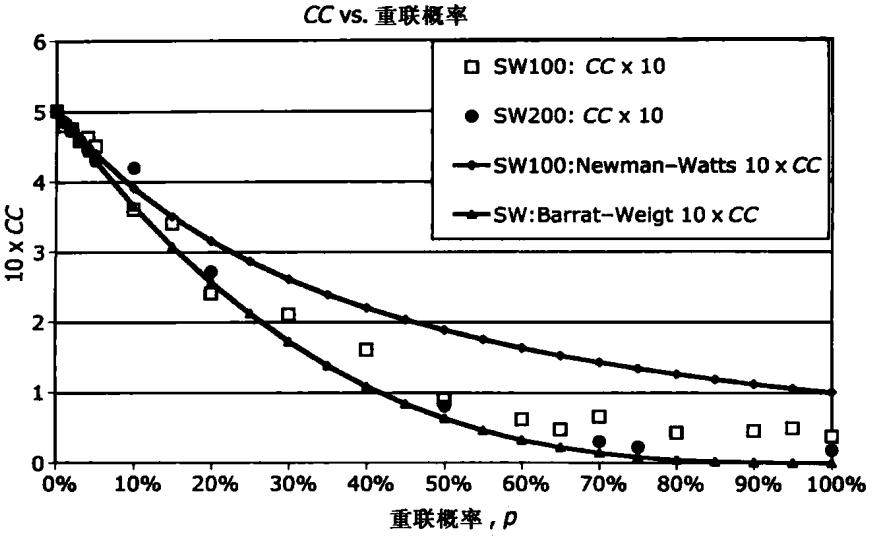


图 5-6 WS 小世界的聚类系数与重联概率图，对于 $n = 100,200$ 和 $m = 200,400$（$\lambda = 4, k = 2$）

在图 5-6 中，Barrat-Weigt 近似可以更好地拟合数据，但是会低估了图 5-5 中仿真的聚类系数。Barrat-Weigt 近似很容易理解。$CC(0)$ 是底层 k - 规则网络的聚类系数，因此所有节点最初的聚类系数为 $CC(0)$。当 $p > 0$ 时，就以概率 p 破坏三角，而以概率 $(1-p)$ 保持原状不变。以概率 $(1-p)$ 不重联链路。所有三条链路不重联的概率为 $(1-p)^3$。所有三条链路必须重联，否则三角子图就会被破坏。因此重联必须失败 3 次。失败三次的概率为 $(1-p)^3$。也可以说三角子图以概率 $(1-p)^3$ 保持不变。

例如，假设大小为 $n = 100, m = 400$ 的小世界是由最初的 k - 规则网络然后以概率 $p = 10\%$ 重联构成的。那么期望的聚类系数是多少？最初的 k - 规则网络每个节点具有 $\lambda = 8$ 条链路，这就意味着 $k = 4$：

$$CC(0) = 3\frac{4-1}{2(8-1)} = \frac{9}{14} = 0.643 = 64.3\%$$

$$(1-p)^3 = (1-0.1)^3 = 0.9^3 = 0.73$$

$$CC(4, 0.1) = 0.643(0.73) = 47\%$$

聚类随着熵的增加而减少，因为聚类是一种结构（有序），而重联链路是一种随机性（无序）。增大重联概率就增大了无序，也就降低了聚类。熵与 k - 规则网络的最初顺序相反。

接下来保持重联概率不变而让密度变化。聚类系数会随着密度的提高而缓慢提高，因为提高密度会使网络更加接近完全网络。我们已知完全网络的聚类系数为 1.0，因此聚类系数随着密度接近 100% 而渐近 1.0。注意聚类系数增加的限度：$0.5 \leqslant CC(\text{density}) \leqslant 1.0$，这与 Barrat-Weigt 近似相反。为什么？

考虑图 5-4 中所示的聚类系数 - 密度曲线，并假定密度 =20%。对于网络参数 $n = 100$，$p = 4\%$，聚类系数等于多少？

$$k = n\frac{\text{density}}{2} = 100\frac{0.2}{2} = 10,\ CC(0) = 3\frac{10-1}{2(20-1)} = \frac{27}{38} = 71\%$$

$$(1-p)^3 = (1-0.04)^3 = 0.96^3 = 0.885,\ CC(4, 0.1) = 0.71(0.885) = 63\%$$

因此，聚类系数会随着密度的提高而缓慢提高，如图 5-4 所示。原因很显然，对于固定概率 p 来讲，更多的链路意味着开始有更多的三角子图以及相对较少的重联链路。随着密度增加，三角子图数量也会增加，这会导致聚类系数的增加。在初始 k - 规则网络中的 k 值很大，因此

$CC(0)$ 很大。根据密度找到该关系的一个近似方程，这留给读者作为练习。

Albert 和 Barabasi 总结了很多具有高聚类系数的实际网络（Albert，2002）。例如，由同一电影中的电影演员形成的社会网络的聚类系数为0.79；学术论文合作网络的 $CC = 0.72$；大肠杆菌有机物化学反应网络的 $CC = 0.59$；英语词库中连接同义词形成的网络的 $CC = 0.7$。在社会网络分析中，高聚类意味着朋友的朋友连接到一起。按照网络科学的说法是“物以类聚”。

5.2.5 小世界中的紧度

直观上来讲，小世界中节点的平均紧度应该符合随机网络中紧度的模式，特别是当重联概率增加时。但是结果证明是错误的，由于 WS 小世界生成过程是以 k - 规则网络作为其基础。随着密度增加，小世界变得更加规则，并渐近一个完全网络，就像随机网络一样。但是迁移并非平滑的，如图 5-7 所示。事实上，迁移显著地按照 Z 形（lazy-Z-shaped）曲线进行——这与随机网络和小世界网络的平均紧度相比完全出乎预料。小世界网络的聚类系数在接近 50% 密度时经历了非线性变化。

紧度随着密度的增加而增加，到达某一点，然后向下再向上，随之网络开始更像 k - 规则网络而非小世界网络。向下再向上之后，重新变得有规律性，紧度再次提高直到接近 100% 峰值为止。像第 4 章一样通过进行平均场分析，获得非线性行为的非常粗糙的近似。具体策略如下：

1. 修改前一章中获得的平均场近似，$\text{closeness(random)} = O((1 - \text{density})z)$ 以便容纳 k - 规则性对网络的影响，其中 $z = \lambda^r$，λ = 平均度，$r(\text{random}) = O(\log(n)/\log(\lambda))$：

a. 用密度替代（1 - density），因为小世界随着密度增加而增加紧度——最多到达某一点（50%）。

b. 注意从小世界到 k - 规则网络的转变大约在 $r = 1$ 处进行。

c. 由于紧度定义的方式，利用有向路径长度 1 贡献零紧度的事实。

2. 从（使用程序 Network. jar）仿真所收集的数据点中，估计曲线拟合参数 C_1 和 C_2。

任意密度的小世界网络可以通过首先构造一个 k - 规则格子产生。例如，如果 $n = 100$，$m = 100$，$k = 10$。那么就有百分比为 p 的链路随机重联。这种随机重联注入了通过网络的捷径，这就导致任意其他对节点之间路径节点数量的明显减少。重联 pm 链路的效应被建模为路径数的减少，如下所示：

$$r = \frac{A \log_2(n - (1 - p)\lambda)}{\log_2(\lambda)}$$

这里 p = 重联概率，A = 常数。通过典型节点的路径的平均场数与 z 成正比，$z = \lambda^r$。

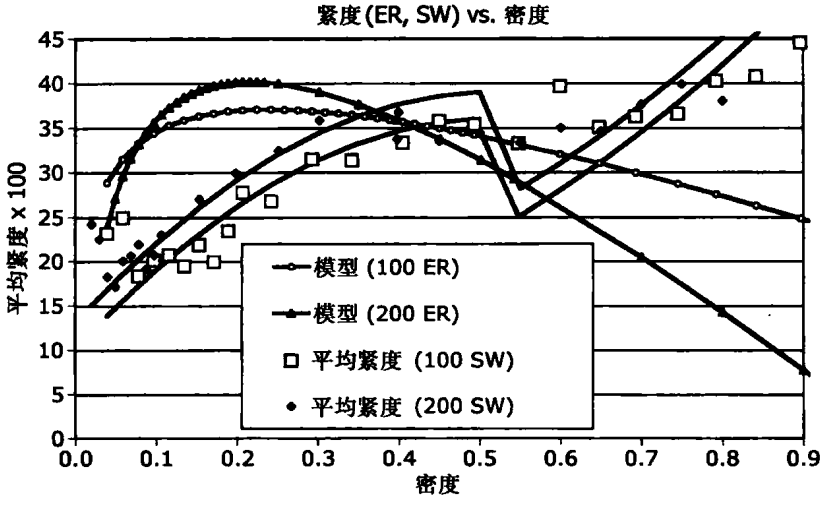


图 5-7 $n = 100,200$ 的小世界的平均紧度与密度，与规模相同的同等随机网络对比

当 $r < 1$ 时，只考虑紧度，小世界就不再像小世界的行为了，而是变得更像 k - 规则网络。因此，当 $r < 1$ 时，我们为 z 建立不同的模型。事实上，较稠密的小世界更像 $k = \lambda/2$ 的 k - 规则网络。因此仅有 $k = \lambda/2$ 条路径通过典型的节点：$z = k = \frac{\lambda}{2}$

将这两种效应结合起来，我们就得到：

$$z = \begin{cases} \lambda^r & r \geqslant 1 \\ \frac{\lambda}{2} & \text{其他} \end{cases}$$

这种迁移效应显示在图 5-7 中的数据和模型中，对于重联概率 $p = 4\%$，则发生在大约 50% 密度处。但是近似是非常粗略的：

$$100(\text{closeness}(\text{small world}, p)) = C_1(\text{density})z + C_2$$

$$z = \begin{cases} \lambda^r & r \geqslant 1 \\ \frac{\lambda}{2} & \text{其他} \end{cases}$$

$$r = \frac{A\log_2(n - (1-p)\lambda)}{\log_2(\lambda)}$$

$$n = 100 \text{ 和 } p = 4\% : A = 1, C_1 = 1, C_2 = 10$$

$$n = 200 \text{ 和 } p = 4\% : : A = 1, C_1 = 0.5, C_2 = 13$$

随机网络和小世界网络的紧度之间存在着很大的不同。随机网络在密度接近 100% 之前没有规律性。平均紧度主要为平均路径长度和稀疏性的函数。另一方面，小世界网络具有大量与密度无关的规则性——随着密度的增加规则性对平均紧度的影响会增加。在大约 50% 左右到达“临界点”，这就导致小世界行为更像 k - 规则网络而非随机网络。这种规则性在决定紧度时成为最重要的因素。一般来讲，超过 ~20% 密度，聚类倾向于增加平均紧度，而随机性倾向于减少平均紧度。当密度刚好为 100% 时，图 5-7 中的完全网络端点的平均紧度等于多少？这留给读者作为练习。

5.3 相变

在 20 世纪 90 年代后期，在 Stanley Milgram 的社会网络分析 30 年后，Watts 和 Strogatz（Watts，1998）重新激发起对小世界行为的科学兴趣。最重要的是，Strogatz 和 Watts 使用小世界网络作为物理世界现象的模型。特别是，Watts 使用由小世界网络行为解释过的同步的思想——一种我们将要在随后的章节中讨论的话题。在本章中我们将研究两种其他行为——相变和网络导航。

物理世界中的相变是在物质从固体状态变成液体、从液体到气体等时发生的。例如，Ising 效应用来解释当惰性铁转变成磁铁时发生的相变。研究思想与下述内容有关：随着重联概率的提高，从 2 - 规则突然转变成随机网络。Watts 讨论了相变是小世界的特性，并且也是不同领域中所观测到现象的同一基础。

5.3.1 路径长度和相变

小世界网络的结构会随着重联概率的增加，从规则变成半规则，再到半随机，最后到完全的随机拓扑。该现象不仅发生在小世界网络中，并且也是随机网络的属性。但是由于随机网络 $p = 100\%$，当发生突然变迁时它就不那么明显。在如 k - 规则小世界的可调整系统中，从规则到半随机的迁移是非常突然的，对于非常小的重联概率平均路径长度快速减少（参见图 5-5b），大约 $p = 0.1\%$。

对于非常小的重联概率增加，网络半径和直径也会降低，并随着小世界迁移到随机网络而继续降低。在图5-5b中，从“大多数垂直”线到“大多数水平”线的迁移被称为相变点，对应于相变的重联概率被称为*转换点* p^*，又称为*相变阈值*（Barthelewy，1999）。阈值明显地标示出从垂直到水平的交界点，并标记为从“大多数 k -规则”到“大多数随机”拓扑的迁移。

将 p^* 定义为图5-5b中平均路径长度 L 与 p 的斜度等于 -45° 的点。这对应于更改速率 $\delta L/\delta r = -1$：

$$L = \frac{n/4k}{(r)^q}, \frac{\delta L}{\delta r} = -1 = -\frac{nq}{4kr^{q+1}}$$

求解 r，得到：

$$r = \left(\frac{nq}{4k}\right)^{1/(q+1)} = pm, \text{因此在转换点处}, p^* = \frac{r}{m}$$

$$p^* = \frac{r}{m} = \left(\frac{nq/m}{4k}\right)^{1/(q+1)}$$

在图5-5b中，$q = \frac{1}{3}$，$1/(q+1) = 0.75$，$n = 500$，$m = 1000$，$k = 2$，因此转换点 p^* 为0.975%：

$$r = \left(\frac{nq}{4k}\right)^{1/(q+1)} = \left(\frac{500\left(\frac{1}{3}\right)}{4(2)}\right)^{0.75} = 9.75$$

$$p^* = \frac{r}{m} = \frac{9.75}{1000} = 0.00975 = 0.975\%$$

因此转换点接近1%。查看图5-5b，很容易看出平均路径的降低大部分发生在1%之前。可见，从 $p = 0$ 到 $p = 1\%$ 的下降负责对应80%的平均路径下降。Barrat和Weigt宣称对于相似的小世界 $p^* \sim (1/n)$（Barrat，2000），产生 $\frac{1}{500} = 0.2\% = p^*$。这种估计小很多，负责对应64%的下降，要比作者的近似公式小16%，但是仍占大多数的下降。

可见，Barrat-Weigt近似更具有意义。重联1000条链路中的0.2%意味着仅有两条链路重导向，而重联1%意味着有10条链路重定向。两条重联链路可能造成路径长度下降到1/4以下，这就意味着路径长度差不多下降 $(125 - \frac{125}{4})/125 = 75\%$。对于图5-5b中的网络，Barrat-Weigt近似就会更加与粗略估计一致。

5.3.2 材料中的相变

相变现象对物理学家很有吸引力，因为它可以解释各种实际效应，例如“液体”突然转变成固体，或非磁铁“突然”变成磁铁——就是所谓的Ising模型[⊖]。这种接近转换点的突变也可以解释生物学中的其他现象（神经建模）和消费者行为（时尚流行和突然“变得流行”的产品）。正因为此，相变值需要更加详细地探讨。

Albert和Barabasi（Albert，2002）提供了物理学和生物科学中关于迁移的简洁研究。*渗流理论*或许是实际中相变的最突出的例子。这种思想很简单，从可能包含多个组件的随机图开始，重联链路，并观察是否以及何时形成巨大组件。从网络上来讲，这被建模成随着重联的增加巨大组件“突然”形成。在渗流理论中，p^* 称为*渗流阈值*。

⊖ Ernst Ising（1900—1998）于1924年创建了Ising模型以便解释磁性。自此之后，它便成了统计物理学中的基本模型。

注意网络不一定是小世界。实际上，渗流理论假设一种均匀的随机网络，因此 p 和密度通过 $pm = n(n-1)(\text{density}/2)$ 关联起来。以固定速率添加链路也会以固定速率增加密度。在某点，网络足够稠密形成一个巨大的组件。

*键渗流*建模成具有 pm 条链路的二维格子。链路受平面 2D 表面约束，这样每个节点可以仅沿着 NEWS ⊖方向连接；因此 m 受限于 $4n$，那么 $pm \leqslant 4n$（密度）。从 $p = 0$ 开始，p 增加直到出现了巨大的组件为止。如果可以从任何节点到达每个其他节点，那么整个网络是一个巨大的组件。

材料也以其他方式从结构化的迁移到随机的。考虑完全连接的具有 $m = n((n-1)/2)$ 条链路的完全网络。这个巨大的连接的网络代表凝固的水立方或完全磁化的金属。现在，随机地选择链路并以概率 $(1-p)$ 删除链路。测试网络并看它是否还是一个巨大的组件。重复随机网络的删除过程直到巨大的网络不再连接为止。在该点，仍会留下 pm 条链路。p^* 现在有何意义？一个渗流网络当 $p < p^*$ 时处于它的次关键阶段，当 $p > p^*$ 时处于超关键阶段。在某种受限情况下（Cayley 图），$p^* = 1/n$。在该阈值之下，没有巨大的组件形成；在该阈值时，形成大小为 $n^{2/3}$ 的巨大的聚类；该阈值之上，复杂的巨大的组件以环形构成。液体“结晶”成固体以及从非磁铁向磁铁迁移对应于对齐链路构成了巨大的组件。这样一来，p^* 就是液相和固相或非磁相和磁相转变的*临界点*。

渗流也可以在如互联网的人工网络中观察到。Broder 等人分析了互联网的 Web 图上的两亿多台服务器，并发现了它的巨大组件，称为*巨大的强连通的组件*（GSCC）（Broder，2000）。互联网的 Web 图是一种“边缘重叠”（bowtie）形状——4400 万个节点（21%）占绝大多数的进入节点发送如电子邮件的信息；另外 4400 万个节点（21%）占绝大多数外出节点接收信息。更大的聚类，5600 万个节点（27%）构成 GSCC。

按小世界效应来讲（短平均路径长度），GSCC 是一个小世界网络。由 Broder 等人报告的平均路径长度（GSCC 中的 5600 万个节点）仅为 16～20 跳！按照由 Adamic 进行的 WWW 高级测量，其中 GSCC 是一个组件，平均路径长度为 35 跳（Adamic，1999）。在另一个极端，当在互联网的路由器级别测量时，Yook 等人估计平均路径长度等于 9 跳（Yook，2001）。因此，GSCC 估计 16～20 跳大约落在互联网非常高级和非常低级测量的中间。在任何情况下，互联网是小世界效应的一个很有说服力的例子。

如结果证明，互联网拓扑是无标度的而非小世界，尽管它显示出小世界效应。无标度网络平均路径长度按照对数近似而非幂律分布：

$$L(\text{scale-free}) = \frac{\log(n)}{\log(\log(n))} = 8.7 \text{ 跳，对于 } n = 5600 \text{ 万}$$

无标度网络的小路径长度是由于它的 hub——度非常高的节点——而非随机链路充当捷径。GSCC 路径长度是 hub 拓扑序列，而不是随机重联。尽管在 Web 图中域级别的聚类是中等的，互联网聚类要比纯随机网络期望值要高。Yook 等人设置了域名级互联网的聚类系数范围为 0.18～0.30，对比之下同等随机网络的为 0.001（Albert，2002）。我们在下一章研究无标度网络的属性，并显示无标度拓扑对实际网络属性的影响要比小世界效应的更大。

5.4 小世界网络中的导航

*网络导航*是一种搜索过程，据此一条信息仅使用如节点度、向心性或紧度等本地信息找到一条从源到目的地节点的路径。这种搜索路径长度要比平均路径长度大得多，这是因为有关最

⊖ NEWS = north（北）、east（东）、west（西）、south（南）的首字母缩写。

短路径的不完全信息，使得需要采用试错法查找而导致的。但是网络导航是一种重要的话题，因为它在如通信等很多领域中建模了很多过程。实际上，小世界中的网络导航可以解释大的通信系统是如何依赖于搜索效率而工作（或不工作）的。

总的来讲，互联网和通信系统需要找到通过复杂网络的路径以传输信息分组，很像 Stanley Milgram 发出的信件必须找到一条从发送者到接收者的路径。这种网络是如何路由分组的，通信网络结构会导致效率的不同吗？详细讲来，小世界拓扑会提高或推迟网络导航吗？

在实际的自然和人造系统中，小世界结构通常是有益于高聚类、低路径长度或两者同时的一系列微规则。例如通信网络的效率是分隔任意两个节点的跳数的函数。在一个神经网络中，通过使网络直径小而使最大传输时间保持最小。这样一来，通过限制网络直径随着 n 的增加而增长，“随机网络”随着尺寸增加而扩展，这一点很重要。

我们已经演示具有小世界效应的网络能够扩展得很好。随着 n 的增加，设计者仅注入少量的随机性到链路结构中，直径就会急剧降低！这样一来，通过注入少量的随机性，对于给定网络密度的传输延迟似乎是可以控制的。这种推测正确吗？要么熵的增加导致更低的传输效率？

遗憾的是，小世界理论不是针对导航规定的，因为它没有告诉如何找到通过网络的最佳路径。例如，在前一节中没有给出路径长度或紧度近似，指示如何从一个节点导航到另外一个节点。没有显示在没有某种全局路由表的情况下，如何选择到达给定节点的路径。在搜索通过小世界网络的路径时应该使用哪些本地信息，为了确定这些我们采用仿真方法。

假设我们以最有效的方式从节点 v 开始尝试到达节点 w。进一步假设我们仅有沿着导航路径 $v \sim x \sim y \sim \cdots w$ 访问的有关节点的本地信息。我们如何选择链路，如何知道选择的链路是否将引导到正确的目的地？这就是网络导航的问题。

网络导航在对等文件共享中有一些应用。例如，20 世纪 90 年代后期，当音乐文件存储在客户的本地硬盘上时，几个文件共享应用程序找到了进入流行文化的方法，并使用导航软件找到它们。在没有对等网络的全局图的情况下，搜索软件在相邻的服务器上将进行深度优先搜索（DFS），并进行扩展直到定位需要的文件为止。

假定我们具有关于每个节点的本地信息，如度、半径和邻接邻居列表。导航软件就可以使用本地信息指导沿着从 v 到 w 路径上链路和节点的选择。但是哪个本地属性最好？假定建议在每个节点上预存以下属性，我们研究四种搜索算法：

1. 随机——在导航路径上的下一个节点随机地选择。
2. 最大度——具有最高度的邻接节点作为下一次选择。
3. 最小半径——越靠中心的邻接节点作为下一次选择。
4. 最大紧度——具有最大紧度比率的邻接节点作为下一次选择。

目的是当随意地选择 v 和 w 作为源和目的地节点时，确定哪种搜索算法（平均）可以以最小的跳数导航。我们假定网络是强连通的（具有一个组件），因此所有节点都可以从其他所有节点到达。这与网络是否是小世界有关系吗？

随机节点选择算法将充当基线——我们期望它将产生最差的导航性能[⊖]。沿着导航路径访问每个节点时，从邻接节点中随机地选择下一个搜索的节点。直观来讲，结果导航路径将是一条通过网络的随机路径。搜索在到达一个循环或死胡同时就返回，当它找到目的地节点时就中止。

直观地看，度高的节点就更加可能被连接到目的地节点。因此，最大度算法根据度从最高向最低排序邻接节点，然后选择到最高度邻居的链路。如果搜索失败，就尝试次最高度的节点，依次类推，直到搜索过所有邻接节点或找到目的地节点为止。

⊖ 实际上，最大紧度算法要比随机算法差！

直觉上，越靠近中心的节点就“越靠近”目的地节点。就这种意义上讲，“越靠近”意味着节点根据半径排序。邻接节点根据半径排序，并按照从最低到最高排序遍历。如果搜索回溯，就选择并遍历第2大的半径邻居，依次类推，如随机的和最大度搜索。

最后，处于中间或接近于所有其他节点的节点（因为其紧度）应该位于到达目的地节点的路径上。因此，邻接节点根据它们的中间状态、紧度进行排序，并按序尝试：首先是最高紧度值，接着次高紧度值等，直到到达目的地节点或者阻止成功搜索的死路为止。

在所有情况中，因为网络是强连通的，必须最终成功搜索到目的地节点。每种搜索算法被嵌入了深度优先搜索（DFS），在遇到死路时就进行回溯。此外，节点标记为“访问过”以避免陷于循环。最差的导航路径在到达目的地之前遍历网络的所有节点。四种算法除了选择下一个节点的方式不同外，其他都是相同的。

在程序 Network. jar 的随机路径搜索方法中，通过递归地调用来实现 DFS 自身，直到遇到一条死路或遇到循环而找不到目的地节点为止。它依赖于来自类 Shuffler 的一种随机方法（这里没有显示），然后在每一个随机邻接节点上发起 DFS。在遇到死路或循环时，DFS 所沿着的路径中止。最后，一条或多条路径到达目的地节点，导致 reply. found 被设置为 true。

列表 randomizer. card [ ] 暂时保留随机的邻接节点，以便它们被重新“洗牌”然后搜索。如果需要保证搜索整个网络，这是必需的：

```
private NavReply Random_Path(NavReply reply, int dest){
        int DEAD_END = -1;
        Shuffler randomizer = new Shuffler(nLinks);
        if(reply.found || reply.node == DEAD_END) return reply;
        reply.hops++;
        node[reply.node].visited = true;
        int next_node = 0;                          //Successor nodes
        int j = 0;                                  //Number of successors

    for(int e = 0; e < nLinks; e++){          //Find all successors
      if(Link [ e] .head == reply.node) next_node = Link [ e] .tail;
      else if(Link [ e] .tail == reply.node) next_node = Link [ e] .head;
      else continue;                            //Skip non-adjacents
      if(next_node == dest) {
            reply.found = true;
            reply.node = next_node;
            return reply;                                 //Found!
      }
      if(node [ next_node] .visited) continue;  //Skip previous visited
      randomizer.card[j] = next_node;           //New (random) order
      j++;
    }
    //Explore j possible next nodes...
    if(j > 0){
      randomizer.SH_Shuffle(j);         //Randomize links
      for(next_node = 0; next_node < j; next_node++){
            reply.node = randomizer.card [ next_node] ;
            reply = Random_Path(reply, dest);  //Depth First Search
      }
    }
    else {
```

```
            reply.hops-;
            reply.found = false;                    //Dead-end reached
            reply.node = DEAD_END;
        }
        return reply;
    }//Random_Path
```

对象 reply 包含跳数、当前节点以及是否已经找到了目的地节点。最初，reply. node 包含起始节点 v，当找到了目的地节点 w 时，它也保存在 reply. node 中。NavReply 数据结构给出如下：

```
class NavReply {
            int hops = 0;                   //Number hops
            int node = 0;                   //Current node
            boolean found = false;          //Destination found?
        }
```

除了下一个节点选择方法外，最大度、最小半径和最大紧度 DFS 算法相同。代替随机的邻接节点列表，最大度和最大紧度算法根据节点度或紧度按降序排序下一个节点，最小半径算法根据半径按升序排序。再次，使用类 Shuffler 排序，但是没有在这里显示。

```
private NavReply Max_Degree_Path(NavReply reply, int dest){
            int DEAD_END = -1;
            Shuffler sorted = new Shuffler(nLinks);
            if(reply.found || reply.node == DEAD_END) return reply;
            reply.hops++;
            node[reply.node].visited = true;
            int next_node = 0;                                  //Successor
            int j = 0;
            for(int e = 0; e < nLinks; e++){                //Find next node
                  if(Link[e].head == reply.node) next_node = Link[e].tail;
                  else if(Link[e].tail == reply.node) next_node = Link[e].head;
                  else continue;
                  if(next_node == dest) {
                        reply.found = true;
                        reply.node = next_node;
                        return reply;                       //Found!
                  }
                  if(node[next_node].visited) continue;
                  sorted.card[j] = next_node;
                  sorted.key[j] = node[next_node].degree;
                  j++;
            }
            if(j > 0){
                  sorted.SH_Sort(j, false);                 //Descending order
                  for(next_node = 0; next_node < j; next_node++){
                        reply.node = sorted.card[next_node];
                        reply = Max_Degree_Path(reply, dest);
                  }
            }
```

```
        else {
              reply.hops--;
              reply.found = false;
              reply.node = DEAD_END;
        }
        return reply;
  }//Max_Degree_Path
  //Minimum Radius Method
  private NavReply Min_Radius_Path(NavReply reply, int dest){
        int DEAD_END = -1;
        Shuffler sorted = new Shuffler(nLinks);
        if(reply.found || reply.node == DEAD_END) return reply;
        reply.hops++;

  node[reply.node].visited = true;
  int next_node = 0;                                    //Successor
  int j = 0;
  for(int e = 0; e < nLinks; e++){                  //Find next node
        if(Link[e].head == reply.node) next_node = Link[e].tail;
        else if(Link[e].tail == reply.node) next_node = Link[e].head;
        else continue;
        if(next_node == dest) {
              reply.found = true;
              reply.node = next_node;
              return reply;                                 //Found!
        }
        if(node[next_node].visited) continue;
        sorted.card[j] = next_node;
        sorted.key[j] = node[next_node].radius;
        j++;
  }
  if(j > 0){
        sorted.SH_Sort(j, true);                    //Ascending order
        for(next_node = 0; next_node < j; next_node++){
              reply.node = sorted.card[next_node];
             reply = Min_Radius_Path(reply, dest);
        }
  }
  else {
        reply.hops--;
        reply.found = false;
        reply.node = DEAD_END;
  }
  return reply;
}//Min_Radius_Path
```

方法 Radius_Path( )实现其他两种搜索算法：最小半径和最大紧度。在这两种情况下，每个节点的半径字段被用做排序的关键词。但是，最小半径搜索必须以升序排序邻接节点，而最大紧度搜索算法必须降序排序邻接节点。这可以通过布尔参数 ascending 指示，当等于 true 时表示升序，而等于 false 时则为降序。在每种情况下，搜索以一个随机选择的节点传入到方法如 reply 开始，而以 dead_ end 或目的地节点的索引结束。注意深度优先搜索递归地实现：

```
private NavReply Radius_Path(boolean ascending, NavReply reply, int dest){
            int DEAD_END = -1;
            Shuffler sorted = new Shuffler(nLinks);
            if(reply.found || reply.node == DEAD_END) return reply;

      reply.hops++;
      node[reply.node].visited = true;
      int next_node = 0;                                          //Successor
      int j = 0;
      for(int e = 0; e < nLinks; e++){                    //Find next node
            if(Link [ e] .head == reply.node) next_node = Link [ e] .tail;
            else if(Link [ e] .tail == reply.node) next_node = Link [ e] .head;
            else continue;
            if(next_node == dest) {
                  reply.found = true;
                  reply.node = next_node;
                  return reply;                                   //Found!
            }
            if(node [ next_node] .visited) continue;
            sorted.card[j] = next_node;
            sorted.key[j] = node[next_node].radius;
            j++;
      }
      if(j > 0){
            sorted.SH_Sort(j, !ascending);  //true = ascending;
            false = descending
            for(next_node = 0; next_node < j; next_node++){
                  reply.node = sorted.card[next_node];
                  reply = Radius_Path(ascending, reply, dest);
                        //Recursive call
            }
      }
      else {
            reply.hops--;
            reply.found = false;
            reply.node = DEAD_END;
      }
      return reply;
}//Radius_Path
```

哪种算法最佳，与网络结构有关吗？通过研究图 5-8 就可以回答这些问题。令人惊讶的是，随机导航算法并非最差的！对于 2 - 规则 WS 小世界网络，随机要比最大紧度和最小半径好。一般来讲，最大紧度是最差的，因为它需要最多跳数才能到达目的地节点。

接下来，注意在所有情况下（$p < 5\%$ 的小世界除外），最大度算法是最佳导航算法。换句话来讲，当使用最大度算法要比使用其他三种时的平均导航路径长度要小。实际上，从最差（最多跳数）到最好（最少跳数）排行导航算法，结果如下：

1. 最慢：最大紧度。
2. 慢：随机。
3. 中等：最小半径。
4. 最快：最大度。

这种结果可以直接应用于通信网络理论中。例如，称为*开放最短路径优先*（OSPF）的路由方案在互联网上沿着从发送者到接收者路径找到下一台路由器或交换机。OSPF 需要频繁地更新每台路由器或交换机上的全局信息。这里的结果建议，根据度简单地排序邻接的路由器和交换机，这种方法与 OSPF 一样好甚至更优，这留作读者练习之用。

接下来的观察与网络结构有关。根据图 5-8，三类网络中的导航性能排序如下：

1. 慢：具有小重联概率 p 的小世界。
2. 中等：随机网络。
3. 快：无标度网络。

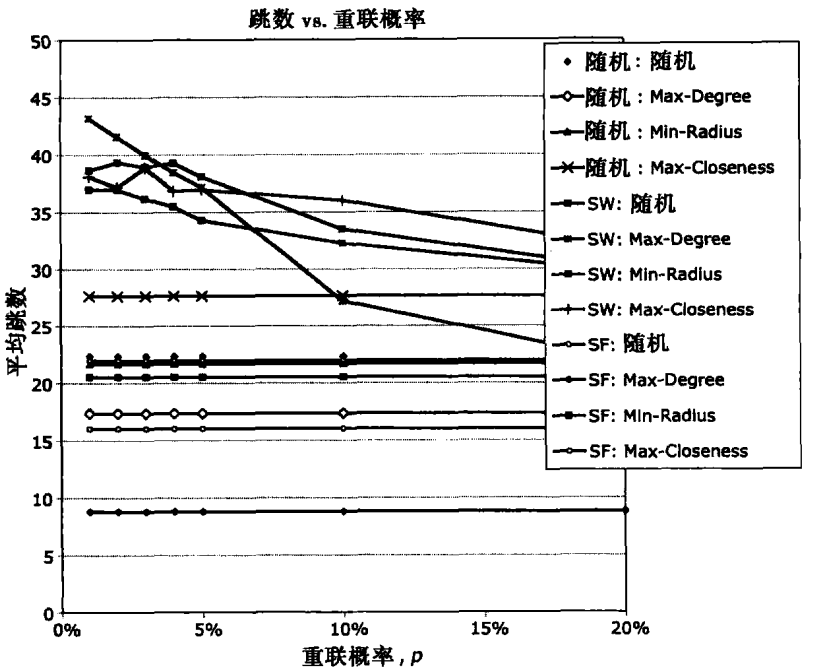


图 5-8 随机、小世界和无标度网络（$n=100$）的平均导航路径长度与重联概率 p，使用四种导航算法：随机、最大度、最小半径和最大紧度

当然，小世界网络随着重联概率的提高变得更像随机网络。最大度搜索算法随着重联的增加快速提高直到它接近随机网络中的最大度搜索性能为止。在图 5-8 中的 SW：Max-Degree 线是所有类型中降低最快的直线。似乎最大度搜索的探索效率是足够的，甚至少量随机性就达到最大值。

无标度网络属性在接下来的一章中详细研究。注意无标度网络比小世界和随机网络导航快得多。最大度算法对于无标度网络特别好，它包含度非常高的节点。再次，该结果在如互联网的通信网络设计中非常重要，因为互联网是一种无标度网络。

奇怪的是，小世界网络类在这种对比中排在最后。小世界仅使用本地信息是很难导航的。小世界网络的两个主要属性是它们的小直径和短特征路径长度。那么为什么会用更大数目的跳数来导航？回顾网络的平均路径长度是基于所有节点对的最短路径。沿着两个节点之间的路径使用有关每个节点的本地信息导航不能保证最短路径。使用本地信息的网络导航找到第一条路径，但不是最短的路径。

在互联网所采用的 OSPF 算法中，交换机和路由器表必须存储沿着最短路径的系列地址（节点）。这张表包含整个网络的全局而不是本地信息。如果使用 OSPF 算法，人们期望得到与本书获得的平均路径长度相一致的结果。但是，路由表必须不断地更新，并且它们不是内容敏感

的——也就是它们不存储可能存储在目的地节点的内容位置。OSPF 路由器表使用索引功能可得到极大地加强，支持根据内容的网络导航。

还有一项值得注意。通信网络必须考虑节点和链路的带宽或容量。这里提供的分析不包括带宽限制，这可以极大地降低 OSPF 和其他导航算法的性能。高度节点也是高负载的节点，因为它们必须处理汇聚到此的所有带宽总和。因此，最大度导航算法可能被由高度（许多链路汇聚到一个节点上）引起的拥塞瓶颈所停滞。高度节点必须也维持高带宽，以有效地在通信网络中导航。一种容纳高度需求的方法是使用冗余——将高度节点分成两台服务器，每台具有一半链路和一半内容容量。

5.5 小世界网络中的弱联系

在前一章中，我们探索了 Flatland Pointville 社会网络，其中平均每个市民知道 24 个其他市民。Pointville 的人口数为 $n = 129$。假定 Pointville 是一个小世界网络，那么何谓跨越 Pointville 的弱联系？回顾*弱联系*是这样一串熟人，即对于任何节点对 $u \sim w$，他们都能穿越整个人群而引导从 u 到达 w。Pointville 的拓扑结构会使弱联系的强度不同吗？

假定在 Pointville 中没有隐士，这种小世界人口的平均路径长度使用 Newman-Moore-Watts 方程计算或者作者的方程计算。在前一章中，随机 Pointville 的直径和半径分别为 2.36 和 2.18。假定 $k = 2$，重联概率为 $p = 6\%$，随机 Pointville 如何与小世界 Pointville 比较？

$$n = 129,\ m = 2(129) = 258,\ r = (2)(0.6)(258) = 30.96$$

$$\beta = \text{sqrt}(30.96^2 + 4(30.96)) = 32.9$$

$$\text{arc\_tanh}\left(\frac{30.9}{32.9}\right) = 1.75,\ \frac{2n}{\beta k} = \frac{258}{32.9(2)} = 3.92$$

$$\text{avg\_path\_length}(\text{Point ville}) = 3.92(1.75) = 6.85 \text{ 跳}$$

使用作者的近似获得的平均路径长度为：

$$\text{avg\_path\_length}(\text{Point ville}, 0.06, 129) = \frac{\frac{129}{4(2)}}{((0.06)(2)(129))^q}$$

假定 $q = \frac{1}{3}$，$\text{avg\_path\_length}(\text{Point ville}, 0.06, 129) = 6.5$ 跳

仿真结果给出了平均值为 6.5 跳，因此“弱联系”的长度近似 6.5 跳。三种方法一致，都在小百分比点之内。与前一章中给出的随机网络方案比较，Pointville 成员在这种小世界网络中要比在随机网络中相互更加隔离。为什么？

解释取决于两种 Pointville 版本的相对密度。在前一章中，随机网络包含 $m = 24(n/2) = 1548$ 条链路，而 2 - 规则小世界 Pointville 包含 $m = 2n = 258$ 条链路。因此彼此的密度有很大的不同（分别为 0.1875 与 0.0313），路径长度随着密度的增加而减少。

容纳同等的密度，让 $k = 12$，因为一个 $n = 129$、$\lambda = 24$ 的 12 - 规则网络具有 1548 条链路。应用 Newman-Moore-Watts 近似，我们就有

$$n = 129,\ k = 12,\ m = 12(129) = 1548,\ r = (2)(0.06)(1548) = 185.8$$

$$\beta = \text{sqrt}(185.8^2 + 4(185)) = 187.75$$

$$\text{arc\_tanh}\left(\frac{185.8}{187.75}\right) = 2.62,\ \frac{2n}{\beta k} = \frac{258}{187.8(12)} = 0.115$$

$$\text{avg\_path\_length}(\text{Point ville}) = 0.115(2.62) = 0.3 \text{ 跳}$$

这次 Newman-Moore-Watts 方程低估了实际值，因为仿真平均路径长度是 2.13 跳，这可以与

随机 Pointville 的路径长度相比较。对于 $k = 12$ 和 $q = \frac{1}{3(2^{\log_2(12)-1})} = 0.055$，使用作者的近似方法，产生 2.09 跳[⊖]。这是非常接近的，但仍低估了实际值。另外一个有益于社会网络分析的属性是紧度。这种 Pointville 版本的平均紧度是多少？使用图 5-7 和由曲线拟合导出的近似方程，我们得到：

$$\lambda = 2\frac{m}{n} = 2\left(\frac{1548}{129}\right) = 24(\text{给定}),\ \text{Density} = \frac{\lambda}{n-1} = \frac{24}{128} = 0.1875$$

$$r = \frac{\log_2(n-(1-p)\lambda)}{\log_2(\lambda)} = \frac{\log_2(129-(0.96)24)}{\log_2(24)} = \frac{6.73}{4.58} = 1.47,\ z = \lambda^r = 24^{1.47} = 106.9$$

$$100(\text{closeness}(\text{Pointville},0.04)) = \text{density}(z) + 10 = 0.1875(106.9) + 10 = 30.0$$

对于 $n = 129$ 和 $4\% \leqslant p \leqslant 5\%$ 的仿真结果，得出平均紧度为 26.9，因此近似值大约为实际值的 10%。但是紧度（随机 Pointville）=42.7 来自前一章的可以比较的分析。如此，小世界 Pointville 比起随机 Pointville 中的中间影响力有很大的不同。为什么？

通过这里的仿真所产生的小世界网络和随机网络的主要不同在于底层的 WS 小世界网络的 k - 规则结构。对于小的重联概率 p 值，k - 规则结构在小世界形状属性中是主导属性。正因为此，读者应该了解通过 WS 生成过程导入到小世界效应的偏差。特别是，不必要使用 k - 规则格子作为小世界的开始。任何稀疏的和规则网络都可以作为开始。但是，如紧度、聚类系数和路径长度的属性将受到影响。

此外，这些结果仅用于这里提供的根据重联概率 $p = 4\%$ 得出的数据。重联概率对熵有很大影响，在小世界中的小世界效应也是如此，因此我们没有额外的分析就不能得出结论。这留给读者作为练习。

5.6 分析

小世界网络类介于结构化网络和随机网络之间。它们的随机性是可扩展的，从非常低的熵到可与随机网络相比较的熵。从某种意义讲可扩展性是可以调整的，它可以先验地设置成重联概率 p，然后通过添加或减去链路来更改密度。小世界网络拓扑包括相对高的聚类和紧度。这些属性是 WS 生成过程中从最初的 k - 规则网络中继承而来。如果低聚类网络中需要小世界效应，那么就从低聚类规则网络如格子或超环形开始重联。如果需要高度的紧度，那么就从高平均紧度网络如超立方开始。进一步对 WS 生成过程的调查留作读者练习之用。

或许小世界网络相对较低的平均路径长度是令人意想不到的，这在于原本我们对它的期望值要大得多。直觉告诉我们，WS 产生的网络的路径长度应该与 2 - 规则网络 $n/8$ 的平均路径长度相似，因为它与最初导出的 2 - 规则网络相似。但是这就忽略了随机网络的长远本质，这会倾向于将整个网络一分为二。结果说明，平均路径长度至少与幂律分布下降一样快。这就是小世界效应的实质。

我们也演示了小世界网络具有不寻常的高聚类系数。但是我们显示了聚类主要从用于 WS 生成过程的作为起点的 k - 规则网络中继承而来。因此，聚类并非小世界效应的固有属性。实际上，随着密度增加（参见图 5-4），聚类系数和路径长度向相反的方向变化。当 k - 规则 WS 网络接近 100% 时，路径长度和聚类系数做何变化？聚类系数会慢慢地上升到 100%，路径长度则快速降低到 1。

⊖ 每次加倍 k 时，就平分 $q = \frac{1}{3}$。因为 k 加倍了 $\log_2(12) - 1$ 次，$q = 0.055$。

聚类系数由底层的 k - 规则网络来确定，而非小世界效应所确定。我们使用超环形网络替代 k - 规则网络加以演示。结果小世界网络假定超环的相似性替代 k - 规则格子。我们通过比较由 WS 算法产生的小世界网络与由 Gilbert 生成过程产生的随机网络，进一步验证该假设。假设 n = 100 、m = 200 、p = 5% 的小世界网络，以及 n = 100 、m = 200（p = 4% ）的同等的 Gilbert 随机网络，两种网络具有相同数量的节点和链路，但是小世界网络是随机重联的，Gilbert 网络也是随机的，但在其初始的完全网络中的 4950 条链路中仅保留 4%（约 200 条链路）不变。WS 过程保留最初 2 - 规则网络的大多数聚类，而 Gilbert 过程则不然。

我们得出结论，小世界聚类是两个因素的结果：初始网络的内在聚类和重联方式。WS 过程保留了一定量的聚类，因为每条重联链路的一端锚定在一个高聚类稀疏的节点上。Gilbert 过程不保留锚定点，因此其最初的完全网络的内在聚类丢失了。

WS 小世界网络与随机网络相比较如何？图 5-9 定位小世界网络类有些对称包含环形、随机和小世界网络图的最初的 Kiviat 图。但是这是一个典型的小世界。我们知道这些属性的每一个都可以调整成匹配需要的值。就这种意义来讲，小世界的 Kiviat 图的形状是可塑的，并且因此是非典型的。因为其 Kiviat 图的对称性，小世界网络类可被认为是最平衡的网络类。

小世界网络可为实际建模。美国西部电网、某些生物系统中的神经网络以及各种社会网络属于小世界网络。这种系统的一个属性是：从任意选择的一个节点向另外一个节点在未知网络的全局信息的情况下传输信息所需要的时间。我们演示了在三种测试中的最佳算法是最大度算法。最大度节点跳导致了最少的跳数，因此用了最少的时间。这种算法简单，通过首先选择最高度的节点它执行整个网络的深度优先搜索（DFS）或直到目的地节点找到为止。在从源到目的地路径上的每个节点，下一个节点的选择是根据度属性排序相邻节点。选择最高度的下一个节点，并且如果 DFS 回溯，选择次最高的，如果 DFS 再次回溯，再选择下一个最高的，依次类推，直到找到了目的地节点为止。

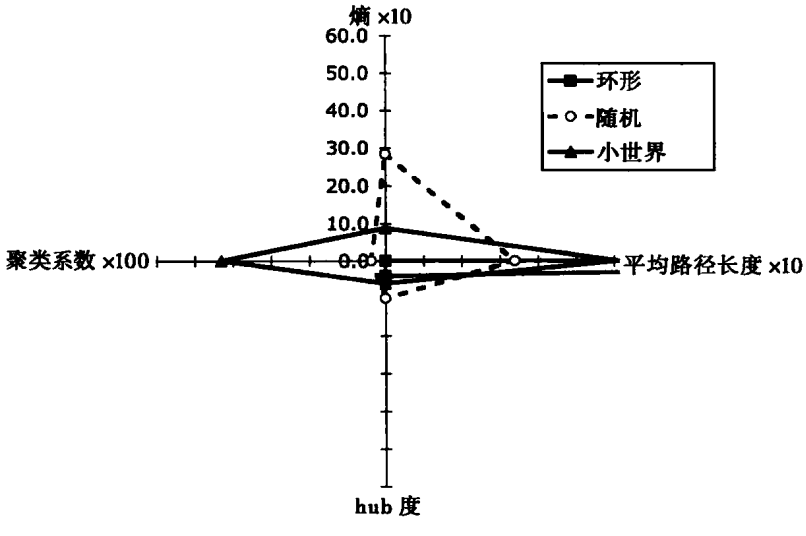


图 5-9 比较环形和小世界网络属性的 Kiviat 图

网络导航具有很多应用，而不仅限于通信网络中的应用。在如互联网中的分组交换系统中，可以使用最大度导航算法研究网络结构本身。这就允许分组利用很少先验信息找到它们自己的通过互联网的路径。这种方法避免了路由器表更新，但是它没有减少网络延迟，因为最大度导航不像最短路径导航那样快。

小世界网络属于非常重要的一类网络，因为它们的扩展性、高度聚类和相对稀疏的特点，可以在物理学中找到很多实现。换句话来讲，衡量链路的经济性和邻居聚类及上述其他属性的小

世界体系结构可以很好地应用于自然界。但是正如我们已经演示的，小世界效应是随机性属性而非其他。此外，WS 生成过程的最初的 k – 规则引入了影响聚类和紧度的偏差。正因为这些和其他原因，WS 小世界失去了意义，更像一个历史古董而非实际中的代表模型。

练习

5.1 修改 WS 生成过程，以便于每个节点保证被连接到小世界网络上。这样就可以保证小世界网络是强连通的吗？

5.2 证明对于小世界网络图 5-3 中的斜度是相同的而与它们的大小 n 无关。

5.3 在初始 5 – 规则网络中，重联 $p = 2\%$ 的链路，具有 $n = 500$ 个节点的小世界网络的平均路径长度和聚类系数是多少？

5.4 OSPF 导航算法与最大度算法有何不同？OSPF 要比最大度更好吗？

5.5 $n = 100$ 和 $p = 64\%$ 的 2 – 规则 WS 小世界网络的熵为多少？在同样数量节点和链路条件下，最高可能的小世界的熵要比随机网络的更大还是更小？

5.6 $n = 100$ 和 $p = 50\%$ 的 2 – 规则 WS 小世界网络的平均紧度是多少？在 $n = 100$ 和 $p = 5\%$ 时又为多少？

5.7 $n = 300$ 的小世界网络的转换点 p^* 是多少？

5.8 当 $p = 10\%$ 时，多大规模（n）的 2 – 规则 WS 小世界网络能够保证具有平均路径长度近似为 6 跳？

5.9 $n = 300$ 和 $p = 5\%$，通过首先访问最大度节点完全导航一个 2 – 规则 WS 小世界网络的期望跳数是多少？将所得结果与 $n = 300$ 和 $m = 600$ 的随机网络中导航的期望跳数进行比较（参考图 5-10）。

5.10 当加倍人口为 $n = 258$ 后，Pointville 的直径为多少？假定认识的平均数保持不变，$\lambda = 24$。

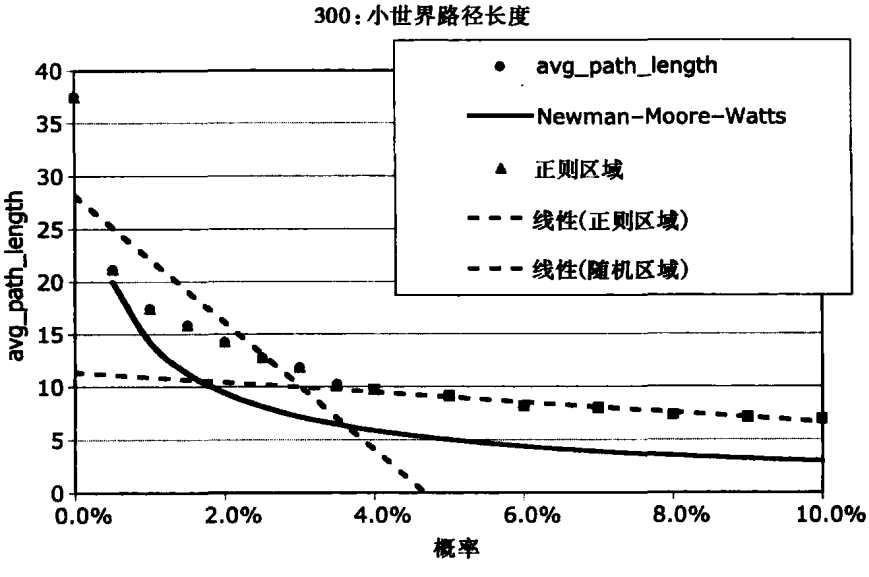


图 5-10 $n = 300$ 的小世界网络路径长度

5.11 构建一个近似 $n = 258$ 和重联概率 $p = 8\%$ 的小世界网络模型。当密度 = 10% 时，这种网络的平均紧度为多少？

5.12 进行一系列实验，确定在幂律分布近似 $n = 200$ 和 $k = 4$ 的小世界网络中的平均路径长度中的指数 q。当 $k = 2$ 时，与作者使用的 $q = \frac{1}{3}$ 相比较。

第6章
Network Science: Theory and Applications

无标度网络

*无标度网络*就是一个具有度序列分布 g'（g'服从幂律分布 $h(k) \sim k^{-q}$）的网络 G，这里 k 是度（$1 < k < \infty$），q 为指数（一般情况下，$2 < q < 3$）。用非数学的语言来描述，无标度网络就是具有少量高度节点和大量低度节点构成的网络。具有高度的少量节点称为 hub，因此无标度网络就是带有 hub 的网络，这将会导致偏态度序列分布。

无标度（SF）网络是非随机的，但是它们的位置比规则或小世界网络（具有小的重联概率 p）更加接近于网络频谱中的随机末端。这是由于它们的熵通常要比规则或小世界网络的大而比纯随机网络要小的缘故。表 2-3 列出了随机、无标度、小世界和规则网络具有的熵值分别为：2.9、2.3、0.9 和 0.0 比特[⊖]。根据密度，无标度网络的熵值可以接近随机网络。因此，我们必须限定在网络科学中所考虑的“同等”网络。

但是无标度网络展示了不与其他网络共享的属性，如 hub——具有非常高的度节点，并能够自由扩展网络密度同时在它的度序列中保持幂律分布关系。在控制 hub 结构的同时可以通过控制密度来控制无标度网络的路径长度。当设计一个具有中心、低导航路径长度和可调节费用（链路数）的网络时就会非常方便。

随着 Albert、Jeong 和 Barabasi（Albert，1999）的先驱性的工作，无标度网络在网络科学中很快上升到显著地位。在此之前，大多数网络被认为是随机的，因此具有服从泊松分布的度序列分布。随后，许多实际网络已经被证明是服从幂律度序列分布而不是二项式分布或泊松分布。例如，许多关键的基础设施系统（如互联网、铁路和天然气/石油系统）已经证明是无标度的（Lewis，2006）。

幂律分布不是指数分布，即使看起来两者很相似。指数分布的尾部消失得非常快。正因为此，幂律分布常常被称为“厚尾分布”。图 6-1 显示了两者之间的显著差异。幂律与 Zipf，Pareto 和 Yule-Simon 律有关，但是更加通用。Zipf 律表明，在英文中单词 w 出现的频率与他在频率表中的秩成反比：$f(w) = c/(r(w))$，这里 $r(w)$ 是单词的秩，c 为常数。Pareto 分布又称为 80/20 法则，也就是说 20% 的人具有 80% 的财富。Yule-Simon 分布近似于幂律分布：$f(k) = 1/k^{\rho+1}$。Yule 可能是第一个证明自然过程服从幂律的现代科学家（Yule，1925）。

在本章中，我们将学习以下内容：

1. Barabasi 和 Albert（BA）（Barabasi，1999）所建议的生成过程，通过*偏好连接过程*生成一个*规范的无标度网络*。从 $n = 3$ 节点开始，余下来的 $(n-3)$ 个节点中的每一个通过添加 Δm 条链路附加到已经存在的节点上而被连接到网络上。对于足够大的网络和足够小的 Δm，m 近似于 $(\Delta m)n$，并且度序列分布服从 $h(k) = k^{-3}$ 形式的幂律。

⊖ 这些数据是通过以非常低的小世界重联概率运行 Network. jar 获得的。

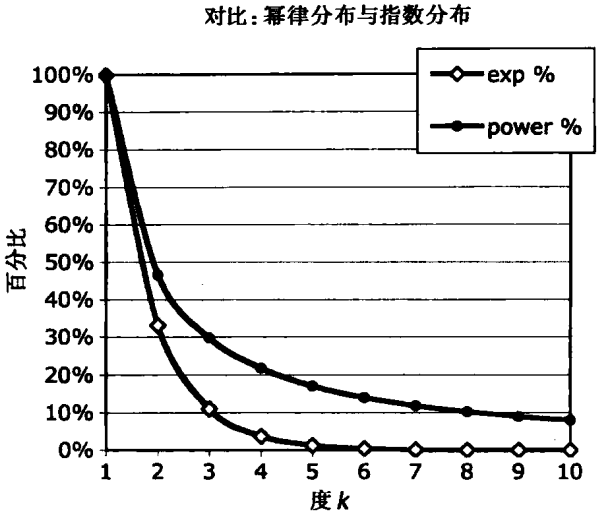


图 6-1 $y = 3\exp(-qk)$ 与 $y = 1/K^q$ 的对比，其中 $q = 1.1$，显示了幂律分布的厚尾

2. 无标度网络的 hub 是一个具有最大度的节点。hub 度与密度的对数成比例：$\text{degree(hub)} \sim O(\log_2(\text{density}))$。

3. 规范的 BA 无标度网络的密度依赖于偏好连接参数 Δm，这是一个将每一个新的节点连接到其他节点构造过程中使用的链路数量。该参数决定了在规范的无标度网络中链路的数量

$$m = (\Delta m)n - \Delta m\frac{(\Delta m + 1)}{2}$$

因此它的密度为

$$d = 2\frac{m}{n(n-1)} = \Delta m\frac{2n - (\Delta m + 1)}{n(n-1)} \sim 2\frac{\Delta m}{n}$$

假定 $n \gg 1$，$\Delta m \ll n$。

4. 规范的无标度网络的度序列分布属于幂律分布：

$$h(k) = 2\Delta m\frac{\Delta m + 1}{(k+2)(k+1)k}$$

或者以大写 O 标记成 $O(k^{-3})$。因为参数 Δm 决定了 BA 网络的密度，度序列也可以表示成密度 d 的形式：$h(k) \sim (n^2d^2)/2k^3$。

5. 规范的无标度网络的熵接近于

$$I(\text{scale-free}) = (\Delta m + 1)\frac{1.1645 - \log_2\left(\frac{\Delta m + 1}{\Delta m^2}\right)}{\Delta m}, \quad 2 \leqslant \Delta m \leqslant (n-1)$$

但是在接近 100% 密度时熵骤然跌至 0，因为网络变成完全连通的网络而不再是无标度网络。

6. 稀疏 BA 无标度网络的平均路径长度低于稀疏随机网络的平均路径长度。我们将小网络的平均路径长度变化表示成

$$O\left(\frac{\log(n)}{\log(n) + \log(\text{density})}\right)$$

密度大于 4% ~ 5%，但是小于 20%。Bollobas 和 Riordan 宣称使用双对数近似可以更好地近似大的网络（如互联网）：

$$\text{avg\_path\_length(scale-free)} = O\left(\frac{\log(n)}{\log(\log(n))}\right)$$

我们也可以说明平均路径长度随着 hub 度线性地降低：

$$\text{avg\_path\_length} = A - 2\frac{\text{hub\_degree}}{300}$$

对于 $n = 200$，或者一般来讲，平均路径长度以 $O(\text{hub\_degree})$ 降低。

7. 无标度网络的平均紧度由其密度和最大的 hub 来决定：

$$\text{closeness(scale-free)} = O(\text{density}(z))\text{，这里 } z = \lambda^r\text{，} r = \frac{\log_2(n - \text{hub\_degree})}{\log_2(\lambda)}$$

因为 hub 趋向于直接连接到大多数其他节点上，它趋向于将 hub_degree 节点从作为中间人的考虑中删除。按照平均紧度，网络类排列如下：随机网络（最高），小世界网络（中等），以及无标度网络（最低）。

8. 无标度网络的聚类系数一般与 n 成反比，随着密度线性地增长：$CC(\text{scale-free}) \sim O(\text{density})$。Barabasi 宣称对于 BA 生成的无标度网络来讲不存在聚类系数解析近似解，但是建议它服从幂律分布：$CC(\text{scale-free}) = O(n^{-0.75})$。因为无标度网络的密度近似于 $d = 2(\Delta m/n)$，聚类系数也会随着 Δm 的增长而增加：$CC(\text{scale-free}) \sim O(\Delta m)$。单个节点的聚类系数与稀疏无标度网络中节点的度成反比；hub 具有低的聚类系数，而非 hub 具有较高的聚类系数。对于较低值的 Δm 来说，聚类系数受限于 $\Delta m/(\text{degree} - \Delta m)$。

9. 无标度网络的导航时间（平均路径长度）使用最大度算法缩减成 $O(1/\text{density})$ 或 $O(\exp(-\text{hub\_degree}))$。但是平均路径长度（导航时间）的大小要远远大于其最短路径平均路径长度。hub 越大，在使用最大度算法时搜索得就越快。

6.1 生成一个无标度网络

Barabasi 和 Albert（Barabasi，1999）研究了度分布为 $h(k) \sim k^{-q}$ 的一类网络，这就意味着许多节点具有低度，少数节点具有高度。具有最高度的节点称为网络的 hub。具有高度的节点也常称为 hub 或次 hub。我们将演示如何构造这种网络，并且从 Newman 和其他人（Newman，2003b）建议的模型中导出度分布函数。

满足 $h(\alpha k) = \beta h(k)$ 的任何函数，对于常数 α 和 β 来讲，被认为是*无标度的*，因为 x 轴（独立变量）放大了一个常数 α，导致也将 y 轴放大了常数因子 β。在这种情况下，$\beta = \alpha^{-q}$，这是一个相对于 k 的常数。独立变量放大一个常数因子的效果就是垂直地向上或向下移动幂律分布曲线，但是不会改变曲线的形状和范围（参见图 6-1）。

无标度网络也是不常见的，因为它们的幂律概率分布具有一个“厚尾”，这就意味着它比物理学中常见的建模成指数分布的现象减少得更缓慢。$h(k)$ 中的减少与 $\exp(-qk)$ 中的减少对比参见图 6-1。特别地，对于大多数 $1 < q < 3$ 的值来说，$k^{-q} > \exp(-qk)$。即使调整幂律函数使得 $h(1) = \exp(-q)$，随着 k 的增加，指数函数也要比 $h(k)$ 减少得更快。这就是为什么幂律分布有时又称为“长尾”或“厚尾”分布的原因了。

Barabasi（Barabasi，2001，2002a，2002b）以及其他人（Newman，2003b；Adamic，2000）广泛地研究了具有厚尾分布的网络，在物理和生物科学中发现了其理论的许多应用。无标度网络表面上可以为实际中的许多现象建模，这些将会在随后的章节中详细探讨。无标度网络类的主要性质在于 hub 节点有非常高的度，以及这种结构对平均路径长度、导航性能和聚类的影响。

但是我们提醒读者注意，纯的幂律分布并不能总能为实际建模。Laherrere 和 Sornette 建议扩展的指数分布在自然界和经济系统中可能会更加准确（Laherrere，1998）：

$$\text{stretched\_exp}(x) = c\left(\frac{x^{c-1}}{x_0^c}\right)\exp\left(-\left(\frac{x}{x_0}\right)^c\right)$$

一般来讲，指数 c 小于1。当 $c = 1$ 时，扩展指数分布便变成了众所周知的指数分布了。Laherrere 和 Sornette 列举出了很多满足该定律的自然和经济系统：1300 次最大的地震分布，在过去 420 000 年中温度的变化，银河系中的无线电和光线发射，油田的保护范围，美国和法国公司的范围，联合国成员国的大小，最经常被引用的物理学家的引用分布，以及生物灭绝事件的分布。

6.1.1 Barabasi-Albert（BA）网络

Barabasi、Albert 和 Jeong（Barabasi，1999）讨论过随机或小世界网络类不能建模某些实际系统，因为他们假定先设置 n 个节点然后再激活底层的微规则。实际网络中，认为无标度网络是动态的而不是静态的，因此应该建模成增长现象。

例如一直扩展的互联网、一直增长的新领域出版物的目录索引、构建物理基础设施（如大区域的电网）等系统是从少数的节点 n_0 开始，然后通过添加新的节点而长大的。他们从少数节点增大到数百万节点通过称为*偏好连接*的进化过程实现。简而言之，偏好连接是有偏向性的，而非随机的——这就导致了带有 hub 的网络。这些早期的研究人员得出结论某些实际系统的动态特性，当他们进化成非随机半结构化的网络时最好由“不公平”地添加节点和链路的网络表示。

Barabasi 等人注意到随机网络服从一种统一的随机添加规则，由此为一对节点添加一条链路的概率是均匀地随机的。再次，这很可能不符合实际。相反，Barabasi 等人建议*偏好连接规则*，声称用于连接节点对的链路的概率与新节点的度成比例。不再是从均匀的随机分布中取样，而是 Barabasi-Albert（BA）生成过程通过从网络 G 的度序列分布中以时间步 t 取样（命名为 $g'(t)$）创建动态网络。随着新的节点和链路的加入，高度节点获得后续链路的概率持续提高，低度节点获得后续链路的概率持续降低，因此随着时间的流逝，“受欢迎的节点”就会更加流行，不喜欢的节点就会更加不受欢迎。

偏好连接得名于以下事实，新节点添加到一个已有的节点上的概率与现存节点的度成正比。度越高，新链路就越有可能连接到节点上。显然，随着网络的增长，高度节点具有更多的链路，而低度节点具有更少的链路。度序列分布取决于低度，因为增加节点数会降低度当少数节点极大地提高度时。

下面给出的生成过程满足 Barabasi 等（Barabasi，1999）的设计。在以下引用中（Barabasi，et al.，1999，pp. 7-8），“顶点”指代节点，“边”指代链路：

（1）*增长*：从少数（m_0）顶点开始，每个时间步添加一个带有 $m(\leqslant m_0)$ 条边（将连接到系统中已经存在的顶点上）的顶点。

（2）*偏好连接*：在选择要将新的顶点添加到哪个顶点上时，我们假定新的顶点添加到顶点 i 的概率为 P 依赖于顶点的连通性 k_i，$P(k_i) = \dfrac{k_i}{\sum_j k_j}$。

（3）时间步 t 后，模型变成了具有 $N = t + m_0$ 个顶点和 mt 条边。

我们通过设置 $n_0 = m_0 = 3$ 来简化 Barabasi 等的生成过程，并使用 Δm 替代 m 以避免与网络中总的链路数混淆[⊖]。

*规范的 BA 无标度网络*是通过重复应用简单的动态网络构造算法产生的：

1. 从三个节点和三条链路开始，通过每个时间步 t 添加一个节点来增长网络。
2. 使用偏好连接将新的节点链接到 Δm 个其他节点。
3. 重复上述过程，直到所有 n 个节点都添加到网络上为止。

⊖ 像这个三角子图模式是从在生物网络文献中被称为模体（motif）的网络开始的。

这需要采取额外的 $t=(n-3)$ 步才能将额外的 $(n-3)$ 个节点添加到最初的三个节点和三条链路上。在最初的形成阶段中，添加少于 Δm 条链路，因为网络尺寸 n 太小了。当 $n<\Delta m$ 时，仅添加 n 条链路。这样，在增长结束阶段，无标度网络具有：

$$m=(\Delta m)n-\sum_{i=1}^{\Delta m} i=(\Delta m)n-\Delta m\frac{\Delta m+1}{2}$$

下面修改过的 BA 生成过程从一个由三个节点和三条链路组成的“三角网络”开始，然后通过偏好连接将新的节点添加到多达 Δm 个其他节点上。该算法忽略了一些在随后的 Java 实现中解决的一些重要细节。

Barabasi-Albert（修改过的）生成过程

1. 输入：Δm = 添加到每个新节点上的链路数；n = 网络大小。

2. 初始化：将节点指定为 0，1，2，…，$(n-1)$。

给定最大的节点数 $n>3$，最初使用 nNodes = $n_0=3$ 和 nLinks = 3 构建一个完全网络。该完全网络的度序列为 $g=[2,2,2]$，度序列分布为 $g'=[1]$，因为每个节点连接到其他两个节点。

3. 当 nNode≤3 时：

a. 新节点：生成一个新的尾节点 v。

b. 新的链路数：设置 n_links = min(Δm,n)。不能添加比现有节点多的链路。

c. 重复 n_ links 次：

i. *偏好连接*——按度序列累积分布函数 CDF(i) 取样选择一个现有的头节点 u，如下定义：

$$\mathrm{CDF}(i)=\sum_{j}^{i}\frac{\mathrm{degree}(n_j)}{\mathrm{k\_total}},\text{这里 k\_total = 2nNodes}$$

ii. 假设 r 是取自 [0，1) 的均匀统一随机数。那么 u 是取自 CDF(i) 的随机变量：

$$\mathrm{CDF}(u-1)\leqslant r\leqslant \mathrm{CDF}(u);\quad u=r\mathrm{nNodes}$$

iii. 连接 $v\sim u$，注意要避免重复的链路。

偏好连接是无标度网络的主要特色。偏好连接的产生来自网络科学在增长现象中的应用。它是一个在不同研究领域具有不同名字的概念。偏好连接在经济学中称为*报酬增加律*（law of increasing returns），在人工智能中称为*适应性学习*（adaptive learning），在生物学中称为*自然选择*（natural selection）。这就是经常所说的“富者越富”现象，因为高度节点获得更多的链路，这将进一步增加其度。偏好连接是一种正反馈，在广泛或大部分人采纳该商品的基础上加速商品的认知。

*报酬增加*意味着商品随着商品变得更加丰富而变得更有价值。考虑广泛采用的微软 Windows 软件——拷贝越多，新的客户就越有可能购买额外更多的拷贝。兼容性、受欢迎程度和仅在 Windows 上运行的应用程序等因素的组合，是提高桌面操作系统市场占有的一些明显的因素。

在人工智能中的适应性学习是一种仿真生物行为或机器学习的算法技术。每次刺激的响应是通过增加来自刺激的同一响应的概率作为回报。重复增加概率，反过来增加“刺激－响应”对的重复。学习是一种基于由重复和正反馈建立的概率分布的训练。它是一种偏好连接形式。

自然选择是一种相似的过程。在大多数物种中，较强的基因会在后代中被复制，而较弱的基因则衰减下去。所谓的个体*适合函数*类似于幂律分布——个体越适应，就越有可能生存、复制，并将它的基因传递给下一代。随着拟合度的增加，基因就更有可能生存下来，并且随着生存率的增加，因此就会更加适合。该反馈机制通常以非常小的增量重复很多次。随着时间的推移，更适合的个体就会从较弱的祖先中显现出来。

或许偏好连接的最早的定义被 Price 称为*累积优势*（Price，1965，1976），在网络被称为无标度之前就用来解释无标度幂律分布属性。Price 的模型稍微不同于 BA 模型。BA 过程忽视链路方

向，而 Price 模型不会，并且 BA 模型是从头开始，而 Price 模型假定一个已经存在的网络。但是，偏好连接和累积优势是相同的概念。

6.1.2 生成 BA 网络

前面描述的 BA 生成过程忽略的一些 Java 实现的重要细节，将要在下面进行讨论。特别是，Java 方法在需要时找到一种可以替换选择的节点以避免重复链路，并通过从一个演化度序列分布中获取的累积分布函数（CDF）中取样演示如何实现偏好连接。

方法 NW_doCreateScaleFree()解释了上述描述的 BA 过程。由 NW_doAddNode()根据需要创建节点，而 NW_doAddLink()用一条链路连接节点对（当链路复制了现有的链路时除外），并使用 Math_random()生成［0，1）中的均匀的随机数。当根据偏好连接选择的节点导致重复链路时，就特别需要找到替换节点的方法。当这种现象发生时，代码向前滚动变量 j，搜寻具有更高索引的节点。这可以通过增加 j 直到它超过 nNodes 为止，然后再次从节点 0 重新开始。除非网络取得了完备性，该方法保证具有 $m = \Delta mn - 3 - \Delta m((\Delta m + 1)/2)$ 条链路或者密度接近于 $2(\Delta m/n)$ 的无标度网络。

偏好连接是通过从累积分布函数（CDF）的每条新链路中取样头部节点实现的。因为每次添加节点后由于链路总数和节点度的更改，都需要重新计算 CDF。用总度数 $2m$ 进行标准化，以便 CDF 不超过 1.0。详细来讲，偏好连接是通过从 CDF 中取样以便获得变量 j 实现的，它必须落在可能的节点数［0，nNodes－1］范围之内。

```
public void NW_doCreateScaleFree(int delta_m){
    int new_node = 0;          //Select a new node to add to network
    double CDF[] = new double[ nInputNodes];
    nNodes = 0;                //Start from scratch
    NW_doAddNode("0");         //Create initial 3-node network
    NW_doAddNode("1");
    NW_doAddNode("2");
    NW_doAddLink(node[1].name, node[0].name);
    NW_doAddLink(node[2].name, node[0].name);
    NW_doAddLink(node[1].name, node[2].name);
    //Add a node at a time
    while(nNodes < nInputNodes){
      CDF[0] = (float)node[ 0].degree/(2*nLinks); //Initialize
                                                    preferences
      new_node = NW_doAddNode(Long.toString(nNodes));
      int n_links = Math.min(delta_m, nNodes-1); //Delta_m must
                                                   be < nNodes
      for(int m = 1; m <= n_links; m++){     //Connect to n_links
                                               other nodes
          double r = Math.random();          //Sample variate from CDF
          for(int i = 1; i < nNodes; i++) {  //Find preferred nodes
                CDF[i] = CDF[i-1] + (float)node[ i].degree/(2*nLinks);
                int j = 0;                   //Destination node
             if(r < CDF[ 0] || CDF[ i-1] <= r && r < CDF[ i]){
                if(r < CDF[ 0]) j = 0; else j = i;
               //Avoid duplicate links
                while(!NW_doAddLink(node[ new_node].name,
                  node[j].name)){
                  j++; if(j >= nNodes) j = 0;   //Roll forward
                }
                break;                         //Linked!
               }
          }
      }
    }
}//NW_doCreateScaleFree
```

NW_doCreateScaleFree()在每次执行时会生成一个不同的无标度网络。为什么？因为它将新的节点连接到最高度的节点，过程BA似乎是确定的，但它实际却不是。从CDF取样产生一个偏置，但是它不能保证总是选择最高度的节点。这个统计偏差意味着在每次生成一个无标量网络时就产生一个不同的网络拓扑。

这里描述的方法产生了一个比随机网络熵小的网络，因为偏好连接产生一个具有非泊松度序列分布的网络。我们将在接下来的章节中更加详细地由无标度网络类导出幂律分布，并导出封闭形式的无标度网络熵的近似。

6.1.3 无标度网络幂律分布

本节解释Newman的BA生成过程的分析用于产生一个规范的无标度网络（Newman, 2003b）。Newman证明无标度网络度序列分布 $h(k) \sim O(k^{-3})$ 是在添加了每个新节点后期望的度为 k 的节点数的直接更改结果。部分度为 $(k-1)$ 的节点将度提高到 k，另一部分节点将度从 k 提高到 $(k+1)$。部分度为 k 的节点的净变化就是这两者之差。

更详细地讲，我们近似度为 k 的节点中的净更改，通过观察一个新节点经 Δm 条链路连接到网络，这会将部分度为 $(k-1)$ 的节点转变成度为 k 的节点，而将另外一部分度为 k 的节点转变成度为 $(k+1)$ 的节点。添加节点之前和添加节点之后分布的差别被平均化了，这就导致了其解为 $h(k)$ 的*平均场方程*。

由Newman给出的推导做了很多可能不是对所有无标度网络都有效的假设。例如，推导假定是线性的——链路连接的概率直接跟目的地节点的度成正比。换句话来讲，关系也可能与目的地节点度的平方、平方根等成正比。然而，Newman的线性模型似乎给出了BA生成过程的精确解。很多作者使用其他假设给出了类似的结果。例如Barabasi等人（Barabasi, 1999）为这里给出的推导进行了另外一种描述。

我们的幂律分布推导策略如下进行：

1. 在时间步 t 使用 $h(k,t)$ 表示度序列分布，使用 Δm 表示每次添加一个节点时附加的链路数。按每个时间步添加一个新的节点和 Δm 条链路，因此在时间 t 度为 k 的节点数目为 $nh(k,t)$，而在时间步 $(t+1)$ 度为 k 的节点数目为 $(n+1)h(k,t+1)$。

2. 在时间步 t 使用 $h(k,t)$ 表示度序列分布，构建一个*平均场方程*表示度为 k 的节点数量在一个典型的时间步所期望的更改。注意插入一个新节点后期望的度为 k 的节点数量为 $\Delta mh(k-1,t)$，插入一个新节点后不再具有 k 度的节点的期望数量为 $\Delta mh(k,t)$。换句话来讲，某些节点的度从 $(k-1)$ 变成 k，某些节点的度从 k 变成 $(k+1)$。

3. 对于静止的（不随时间变化）的情况下，求解平均场方程，产生 $h(k)$。我们通过从平均场方程中删除 t 来加以实现。时间不变性意味着 $h(k,t+1) = h(k,t) = h(k)$。这就是所谓的*静态过程解*，这就表示无标度网络端点状态。添加过 n 个节点后，$h(k)$ 就是度序列分布函数。

首先，推导出一个连接概率和度序列分布之间线性关系的表达式。在时间步 t，度序列分布 $h(k;t)$ 是部分具有度为 k 的节点。注意 $\sum_k h(k,t) = 1$，独立于 t。偏好连接使用该部分选择度为 k 的以如下概率连接到新节点的节点：

$$\frac{kh(k,t)}{\sum_k kh(k,t)} = \frac{kh(k,t)}{\lambda}$$，这里 λ 表示网络的度

我们也知道对于任何网络 $\lambda = 2(m/n)$，对于BA生成过程 $m = \Delta mn$（近似）。这样，部分具有度为 k 的节点是 $(kh(k,t))/2\Delta m$，将 $\lambda = 2\Delta m$ 代入方程为 $h(k,t)$。现在我们就可以写出度为 k 的节点数变化“前”/“后”的平均场方程。

平均场方程 度为 k 的节点数量的变化等于从度 $(k-1)$ 增加到度 k 的节点数减掉从度 k 增加

到度 $(k+1)$ 的节点数。

平均场方程的左边是在插入新的节点后所期望的度为 k 的节点数，减去插入前的数量：

$$(n+1)h(k,t+1) - nh(k,t)$$

右边是期望的连接到度为 $(k-1)$ 的节点数减去度不再是 k 的节点数，因为它们增加到度为 $(k+1)$：

$$\Delta m(k-1)\frac{h(k-1,t)}{2\Delta m} - \Delta m\frac{kh(k,t)}{2\Delta m}$$

取消掉 Δm 后得到简化：

$$(k-1)\frac{h(k-1,t)}{2} - k\frac{h(k,t)}{2}$$

因此时变平均场方程是：

$$(n+1)h(k,t+1) - nh(k,t) = (k-1)\frac{h(k-1,t)}{2} - k\frac{h(k,t)}{2};\quad k > \Delta m$$

静止的或时间不变的解通过设置 $h(k,t+1) = h(k,t) = h(k)$ 获得，简化成一阶差分方程：

$$h(k) = (k-1)\frac{h(k-1)}{2} - k\frac{h(k)}{2};k > \Delta m$$

差分方程的右边假定 $k > \Delta m$，但是当 $k = \Delta m$ 时，仅有一个节点度从 $(k-1)$ 增加到 k，因此平均场方程限制为 $k = \Delta m, \Delta m+1, \Delta m+2, \cdots$，当 $k = \Delta m$ 时：

$$h(\Delta m) = 1 - \Delta m\frac{h(\Delta m)}{2}$$

求解 $h(\Delta m)$，我们就得到：

$$h(\Delta m) = \frac{2}{\Delta m+2}$$

对于 $k > \Delta m$，产生通解：

$$h(k) = 2\Delta m\frac{\Delta m+1}{(k+2)(k+1)k}$$

对于大的 k 和常数 Δm，$h(k)$ 是 $q=3$ 的幂律分布。事实上，$h(k) = O(k^{-3})$。例如假设 $\Delta m = 2$，幂律分布就为

$$h(k) = \frac{12}{(k+2)(k+1)k}$$

注意这是如何预定义度序列分布的，对于 $k = 2,3,\cdots$，平均来讲，所有由 BA 过程生成的无标度网络，当 $\Delta m = 2$ 时，满足下列分布而与大小 n 无关：

$$h(2) = \frac{12}{(4)(3)(2)} = 50\%$$

$$h(3) = \frac{12}{(5)(4)(3)} = 20\%$$

$$h(4) = \frac{12}{(6)(5)(4)} = 10\%$$

$$\vdots$$

$$h(n) = \frac{12}{(n+2)(n+1)(n)} \sim 0\%;\quad n \gg 1$$

图 6-2 比较理论推导 $h(k)$ 与从上述描述的 Network. jar 和 BA 生成过程获得的实际结果。均方根（RMS）误差对于 $n = 100$ 和 BA 生成过程是可以忽略的。显然，Newman 的线性假设非常适用于 BA 网络。

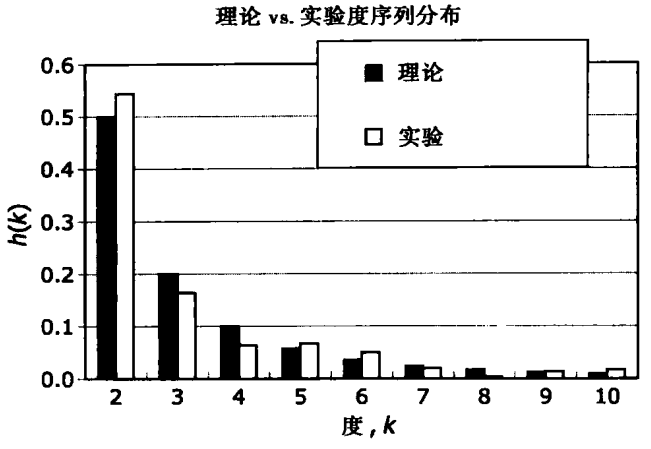


图 6-2 对于 $n = 100$ 和 $\Delta m = 2$ 的无标度网络的度序列分布的理论和实验结果对比。实验结果是三次运行的平均，均方根误差小于 1%

假定 BA 网络的密度：

$$d = 2\frac{m}{n(n-1)} = 2\Delta m \frac{n - 3 - \Delta m \dfrac{\Delta m + 1}{2}}{n(n-1)} \sim 2\frac{\Delta m}{n}$$

我们也可以用密度 d 表示度序列分布函数，将 $\Delta m = n(d/2)$ 代入到上面的方程中：

$$h(k) = 2\Delta m \frac{\Delta m + 1}{(k+2)(k+1)k}$$

$$= nd\frac{nd+2}{2(k+2)(k+1)k} \sim \frac{n^2d^2}{2k^3} \sim O\left(\frac{n^2d^2}{2k^3}\right)$$

该推导为无标度网络产生一个指数 $q = 3$，但是在实际中，如果 $2 \leqslant q \leqslant 3$，网络就被认为是无标度的。Newman（Newman, 2003b）讨论了其他解，根据不同的假设产生了不同的 q 值。从技术上来讲，任何 q 值都满足无标度关系 $h(\alpha k) = \beta h(k)$。

6.2 无标度网络的属性

在本节中，我们将研究由 BA 过程生成的规范无标度网络的属性。除非指定，否则我们使用 $\Delta m = 2$，这就意味着随着网络规模的增加，网络密度降低。这种限制稍后会放宽，但是简化是在保持 Δm 恒定的同时用来探讨熵、平均路径长度和聚类系数的。但是要记住，更改尺寸 n 或 Δm 也会附带更改密度。

首先，我们比较熵、平均路径长度和聚类系数作为网络规模 n（因此网络密度）的函数。这让我们观察 hub 节点度的更改，以及最大度导航的平均距离与尺寸和密度的对比。稍后我们研究保持 n 不变，而通过更改 Δm 更改密度的效应。

下列分析总结在图 6-3 中。通过对不同网络尺寸运行 Network. jar 已经实验性地确定了各种属性。这里显示的属性值是经过放大的，以便在同一图中进行可见的比较，熵和平均路径长度值增大了 10 倍，聚类系数数据增大了 100 倍。

6.2.1 BA 网络熵

从图 6-3 中清楚地显示了随着网络大小变化而导致密度的变化时，熵仍旧保持不变。对于 $\Delta m = 2$，我们获得 I（无标度）~2.4 比特。回想起来，这是很直观的，因为熵 I 由度序列分布确定而独立于尺寸或密度。度序列分布总是幂律分布；因此 I（无标度）是一个偏好连接参数 Δm 的函数。相应地，通过将幂律分布 $h(k)$ 替换到熵方程中可导出一个熵的封闭方程：$I = -\sum h(k)\log_2(h(k))$，积分近似求和：

$$I = \int_{\Delta m}^{\infty} h(k)\log_2(h(k)\delta x = -1.443\int_{\Delta m}^{\infty} h(k)\ln(h(k))\delta x$$

因为 $\log_2(x) = 1.433\ln(x)$，$\ln(x)$ 是自然对数。将 $h(x) = Ax^{-3}$（这里 $A = 2\Delta m(\Delta m + 1)$）代入积分中，并简化对数表示，产生

$$I = -1.443A\int_{\Delta m}^{\infty}\left(\frac{\ln(A) - 3\ln(x)}{x^3}\right)\delta x$$

积分中的第一项很容易积分以便获得 $T_1 = (\ln(A))/(2\Delta m^2)$，第二项积分通过将 u 和 v 代入到积分中得到。

假设 $u = x^{-3}$，$v = \ln(x)$，因此 $\delta u = (-x^2/2)$，$\delta v = x^{-1}$。

积分部分产生积分中的第 2 部分：

$$T_2 = \frac{2\ln(\Delta m) + 1}{4\Delta m^2}$$

将各项结合起来，我们得到：

$$I = -1.433(T_1 + T_2) = -1.433A\frac{\ln(A) - 3\dfrac{\ln(\Delta m) + 1}{2}}{2\Delta m^2}$$

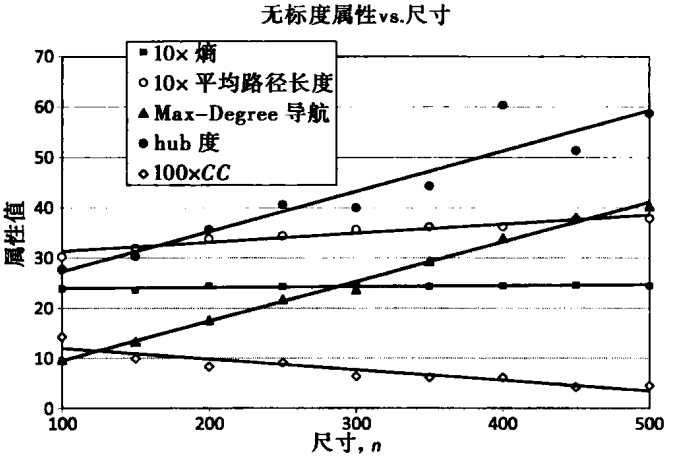


图 6-3 $\Delta m = 2$ 时的 BA 无标度网络属性与尺寸 n 的关系图。注意熵、平均路径长度和聚类系数的比例因子

替代过 $A = 2\Delta m(\Delta m + 1)$ 后，用 $\log_2()$ 替代 $1.433\ln()$，并简化项，我们得到无标度网络熵的分析近似：

$$I(\text{scale-free}) = \Delta m + 1\frac{1.1645 - \log_2\left(\dfrac{\Delta m + 1}{\Delta m^2}\right)}{\Delta m}$$

使用 $I_0 = (\Delta m + 1)/\Delta m$ 可替换成公式

$$I(\text{scale-free}) = I_0(1.1645 + \log_2(\Delta m I_0)); \quad 2 \leqslant \Delta m \leqslant n - 1$$

注意熵仅是 Δm 的函数，而与其他无关。由 BA 过程构建的任何无标度网络的更加具体的熵仅依赖于将新节点连接到网络的链路数，而独立于网络的规模。进一步来讲，熵按照 Δm 的对数增长。例如，$\Delta m = 2$ 的无标度网络的熵为 $I = \frac{3}{2}\left(1.1645 - \log_2\left(\frac{3}{4}\right)\right) = 2.37$。$\Delta m = 5$ 的熵为 $I = \frac{6}{5}\left(1.1645 - \log_2\left(\frac{6}{25}\right)\right) = 3.87$。

熵是密度的间接函数，因为 BA 无标度网络密度随着 Δm 的增加而增加。忽略在前期阶段对偏好连接的限制，链路数 $m \sim \Delta mn$，对于足够大的 n 和足够小的 Δm 就会产生一个近似密度：

$$\text{Density(scale-free)} = 2\frac{m}{n(n-1)} = 2\frac{\Delta m}{n}$$

假定 $n \gg 1$ 而 $\Delta m \ll n$。因此 $\Delta m \sim n$(密度/2)。代入到方程 I(scale-free) 就生成表示成密度函数的熵表达式。这不妨就留作读者练习之用。

无标度网络的熵随着 Δm 的对数函数的增加而增加，直接与密度成比例。与当密度超过50%时变得更加结构化的随机网络不同，无标度熵随着密度的增加而单调地增加，因为到此为止它的度序列分布仍旧是幂律分布的。当密度接近100%时会发生什么？无标度网络就会失效并且它的度序列分布不再是幂律分布了。因此该方程不再有效。为了理解密度对固定尺寸大小的网络的影响，我们在保持 n 不变的同时必须更改 Δm。这种极高密度现象将在第6.4节探索。

6.2.2 hub 度与密度对应关系

极高 hub 度是无标度网络的主要属性。随着可用链路数量的增加，网络的最大 hub 的度也会增加。因此，hub 度会随着密度的增加而增加。但是，这种增加已经减少了反馈，如图 6-4a 所示。

$$\text{Degree(hub)} \sim O(\log_2(\text{density})); \quad 1\% \leqslant \text{density} \leqslant 20\%$$

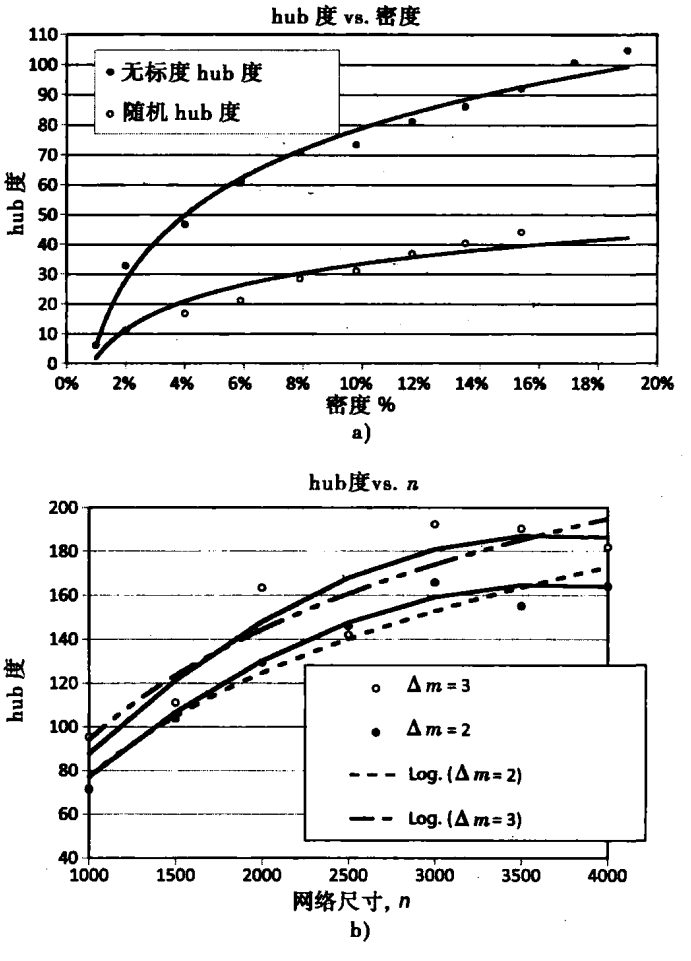


图 6-4 对于固定和可变 n 时网络 hub 度的对比：a) 对于固定 $n = 200$，无标度和随机网络与密度的对比；b) 链路数变量 m，可变密度和尺寸 n，但是固定 Δm 的 hub 度

无论是随机网络还是无标度网络，hub 度都会按密度的对数增长，但是无标度网络 hub 度增长的速率更快。在图 6-4a 中，对于密度为 20% 时，无标度网络是同等随机网络的 3 倍。

图 6-4b 中针对具有可变密度和可变规模 n 的更大的无标度网络画出了最大 hub 度。理论上，我们应该能够估计 hub 度作为 n 的函数，通过求解将 hub 度的更改速率与密度关联起来的简单差分方程：

$$\frac{\delta \text{Degree}(\text{hub},t)}{\delta t} = \text{density} = 2\frac{m}{t(t-1)} = 2\frac{\Delta m}{t}$$

这里我们近似 $(t-1) \sim t$，并替代 $m \sim \Delta mt$，Δm 是 BA 生成过程的输入参数。

通过从 $t = 1$ 到 n 直接积分求解，我们得到

$$\text{Degree}(\text{hub},n) = 2\Delta m \ln(n) = O(\ln(n))$$

图 6-4b 包括对数以及二次曲线拟合从运行多个无标度网络仿真得到的数据，对于 $\Delta m = 2$，3，以及从 $n = 1000$ 到 $n = 4000$ 个节点。点画线显示对数拟合是对仿真中观察的实际 hub 度值的比较差的近似，但是二次曲线拟合会更好（参见实线）。为什么？

二次曲线模型是对无标度网络中的最大 hub 度非常好的近似，因为在非常稀疏网络中最高度节点之间的竞争会从偏爱的节点虹吸掉链路；也就是，随着密度减少（对应尺寸 n 的增加），偏爱使用最高度 hub 以克服链路的短缺。因为这种短缺，链路必须通过在很多高度节点之间的竞争获得。

Barabasi 和其他人已经证明理论上 hub 度是按照图 6-4 中的点画线呈对数增长。但是理论近似假定平均网络从不接近这里所研究的密度极值。例如，在图 6-4b 中以及 $\Delta m = 2$，平均度是四条链路，当 $n = 4000$ 时密度是 0.1%。在某些点，周围没有足够的链路，限制了 hub 度的对数增长。

hub 度更好的近似是下列二次方程：

$$\text{Degree}(\text{hub},n) = n\frac{B-n}{A}$$

对于大的 n，参数 A 和 B 根据经验获得，在图 6-4b 中对于 $\Delta m = 2$，$A = 83\,000$，$B = 7400$；对于 $\Delta m = 3$，$A = 73\,000$，$B = 7400$。这些参数值产生所观察的数据的最小二乘拟合并产生图 6-4b 中的实线。例如考虑 $n = 2500$。对于 $\Delta m = 3$，何谓无标度网络的 hub 度近似？替换到二次近似方程中就产生

$$\text{Degree}(\text{hub},2500) = n\frac{B-n}{A} = 2500\frac{7400-2500}{7300} = 168$$

经验确定的值 142 是通过由程序 Network.jar 产生的三个网络的平均值获得的。$(168-142)/(168+142) = 8\%$ 的近似误差对于图 6-4 中使用的样本大小并不坏。

6.2.3 BA 网络平均路径长度

直观地讲，无标度网络的平均路径长度应该比相同密度的随机网络要短，因为在无标度网络中有更多长距离的链路。但是这种直觉正确吗？本节演示随着密度的增长路径长度逐渐减少——就像它在随机网络中那样。但是无标度网络中的平均路径长度下降在稀疏无标度网络中比在稀疏随机网络中更加显著。这是因为在稀疏无标度网络中 hub 节点要比在随机网络中的度要高得多。无标度网络 hub 在连接任意两个节点时充当一种大部分路径都通过的“中心车站”。实际上，hub 通常是最大的中间节点，因为它们的紧度属性也是最高的。因此对于稀疏无标度网络的直觉是正确的：稀疏无标度网络的平均路径长度要比对等的随机网络的平均路径长度要小。但是随着密度的增加，差距很快消失。

Albert 和 Barabasi 使用平均度 $\lambda = 4$ 跳近似 BA 无标度网络的平均路径长度，仿真数据拟合成对数曲线（Albert，2002）：

$$\text{avg\_path\_length}(\text{scale-free}) = A\log(n+B) + C$$

实际上，当$A = 15$，$B = 100$，$C = -4$时，这是对图6-3中所收集和所显示数据的很好近似。

回顾对于任何网络密度 $= \lambda/n$，如果λ保持不变，那么密度随着尺寸n的增加而减少。因此，因为密度的降低，平均路径长度随着尺寸的增长而逐渐增大。对于这里研究的网络大小和密度，Bollobas和Riordan给出的更精确的估计没有明显提高Albert和Barabasi给出的简单近似(Bollobas, 2001)：

$$\text{avg\_path\_length(scale-free)} = O\left(\frac{\log(n)}{\log(\log(n))}\right)$$

但是Bollobas-Riordan近似对于非常大的网络会更精确。例如，双对数表示对于互联网的特定测量很精确。

遗憾的是，这些理论估计不直接将路径长度与链路数量或hub大小调整联系起来。例如，如果我们保持n不变并改变密度，那么平均路径长度应该与密度的增减相反，密度越高，路径长度越低。为了验证和量化这种假设，我们探讨密度和hub大小对路径长度的影响。

图6-5显示了在同等密度下无标度网络与随机网络平均路径长度的不同。两种网络都倾向于前面第4章中所导出的经验近似，但是无标度网络一般显示出小世界效应——它们具有比同等随机网络稍小的平均路径长度。当密度接近零时，差距就会越显著，显示出的小世界效应在随机网络中要比无标度网络中更加显著。

我们从第4章中得知，随机网络的平均路径长度近似于

$$\text{avg\_path\_length(random)} = \frac{\log(n)}{\log(\lambda)}$$

密度为λ/n，因此我们用n替代λ：

$$\text{avg\_path\_length} = \frac{\log(n)}{\log(n(\text{density}))} = \frac{\log(n)}{\log(n) + \log(\text{density})}$$

但是，无标度网络不是随机网络，因此该近似通过放大分子和分母可以裁剪成非随机拓扑。我们想要找到参数A和C使得下列近似拟合图6-5中的数据：

$$10(\text{avg\_path\_length}) = \frac{A\log(n)}{\log(n) + \log(C(\text{density}))}$$

这里密度以百分比表示：$1\% \leqslant \text{density} \leqslant 100\%$。对于$n = 200$和图6-5中的经验数据，$A = 14$和$C = 2$为无标度网络给出了很好的结果，并且$A = 14$和$C = 1.5$产生的曲线拟合如图6-5所示的随机网络。主要的变化是参数C，它可以改变密度。这就是说在无标度和随机网络对比中使用“不同的”密度。我们将在后面更详细地探讨该问题。

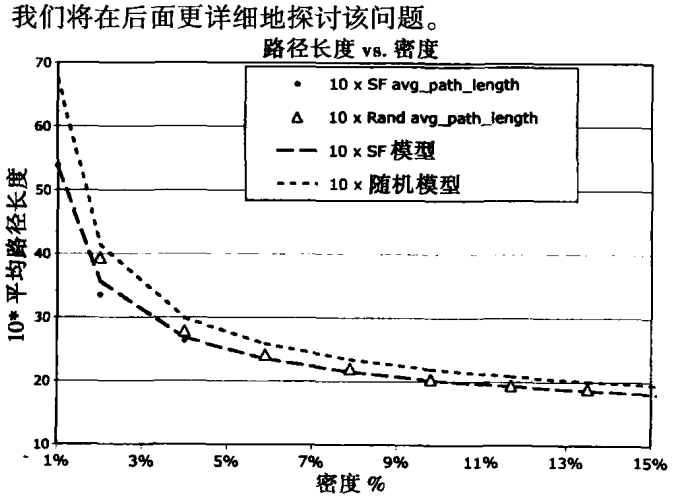


图6-5 BA无标度网络和ER随机网络的平均路径长度与密度对比，其中$n = 200$。无标度网络的平均路径长度要比同等的随机网络的稍低

这种近似准确程度如何？尺寸 $n = 200$ 及密度为4%的BA无标度网络的估计平均路径长度近似于：

$$10(\text{avg\_path\_length}) = \frac{14\log(200)}{\log(200) + \log(2(0.04))}$$

$$= \frac{14(2.30)}{2.30 - 1.1} = \frac{32.2}{1.2} = 26.83$$

$$\text{avg\_path\_length} = \frac{26.83}{10} = 2.683(\text{经验值} = 2.69\ \text{跳})$$

同等随机网络的平均路径长度稍高：2.78 跳。对于不同大小的网络来说，好奇的读者使用程序 Network. jar 可以找到不同的参数 A 和 C 值。

对于密度小于2% ~3%来讲，这种近似不能很好地建模平均路径长度（参见图6-5）。这是因为路径长度在接近相变或转换点处极大地受到小世界效应的影响。当在处理非常稀疏的无标度网络时，hub 的度将是接下来我们将要寻找的平均路径长度的更好的预测参数。

稀疏无标度网络的平均路径长度要比同等的稀疏随机网络的小得多。对于密度小于4%时，在图6-5中是显而易见的。我们假设这种不同是由于与在随机网络中大多数连接节点的度相比在无标度网络中 hub 节点的高度。如果该假设正确，我们就应该能够观察到无标度网络中的 hub 度和路径长度之间的相互关系。直观来讲，随着 hub 度的增加路径长度减少，因为在无标度网络中有更多远距离链路。

图6-6中显示一种稳定的但是杂乱的随着稀疏无标度网络中的 hub 度增加平均路径长度减少的特点。虽然相关性不很强，但是趋向很明显——正如假设猜想的那样，随着 hub 度的增加路径长度趋向下（降低）。这种数据拟合成一条直线，这将产生一种线性关系：

$$\text{avg\_path\_length(BA 网络)} = A - 2\frac{\text{hub\_degree}}{300};\quad A = 3.58$$

该结果支持以下结论：无标度网络的 hub 度越高，平均路径长度越短。hub 导致了也称为小世界的无标度网络。进一步来讲，正如结果证明的，搜索 hub 而不是低度节点，通过无标度网络导航的路径长度是最短的。在其他网络中 hub 度也与平均路径长度相关，但是该命题留作读者的研究项目。

平均路径长度随着（任何）网络密度的增加而降低，因为会有更多可以替换的路径（某些要比其他的短）通过更密的网络。当网络非常稀疏时，节点 u 到节点 w 之间的连接就会有更少的可以替换的路径。在稀疏无标度网络中缺少可以替换的路径，可以部分地但不是完全地由高度 hub 弥补。低度节点更有可能连接到 hub 上，hub 建立远距离链路。hub 的度越高，就越有更多机会使用远距离链路遍历网络。因此，hub 度变成了稀疏网络中的一个更好的平均路径长度的预测参数。hub 是在无标度网络中建立小世界效应的基础。

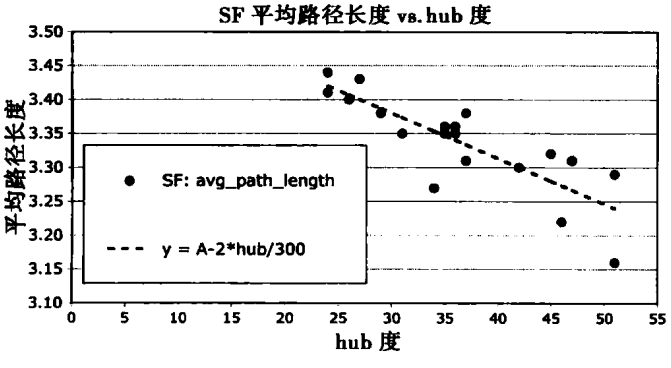


图6-6 在规模为 $n = 200$ 和密度为2%（$\Delta m = 2$）的无标度网络中，平均路径长度随着 hub 度的增加而线性地减少。对这些数据的最小二乘曲线拟合产生 $A = 3.58$ 和斜率等于 $-2/300$

6.2.4 BA 网络紧度

在前面的章节中，我们学习了网络的平均紧度是由网络的密度和拓扑所决定的。一般来讲，密度会增加紧度——直到达某一点——然后由拓扑接管，再确定高密度网络是否增加或减少紧度。例如，随着密度增加由于小世界效应随机网络会经历平均紧度的降低。另一方面，当密度接近100%时，因为它们底层的 k -规则，小世界会增加平均紧度。无标度网络当然是另外一种情况。它们的平均紧度是其 hub 尺寸的直接结果。平均紧度随着 hub 度稳定增加，然后随着密度接近100%而慢慢降低（参见图6-7）。

无标度网络的主导 hub 节点倾向于在它本身和大多数所有其他节点之间插入直接的链路。这种直接连接消除中间节点——没有中间人。这种趋向按照与 hub 度成比例的量缩减紧度。因此，无标度网络中平均紧度的合适模型就是：

$$100(\text{closeness}(\text{scale}-\text{free})) = C_1 \text{densityz} + C_2$$

$$z = \lambda^r$$

$$r = \frac{\log_2(n - \text{hub\_degree})}{\log_2(\lambda)}$$

我们使用简化 $O(\log(\text{density}))$ 模型来确定 hub_degree：$\text{hub\_degree} = A\log_2(B\ \text{density})$。将该模型拟合图6-7中的实验数据便产生如下常数：

$$n = 100: A = 18; B = 25; C_1 = 1, C_2 = 1$$

$$n = 200: A = 36; B = 30; C_1 = 0.5, C_2 = 1$$

例如，在 $n = 100$、$\Delta m = 4$ 的无标度网络中，平均紧度的近似是通过将 Δm 转换成 λ 和密度实现的：

$$m = \Delta mn - \Delta m\frac{\Delta m + 1}{2} = (4)(100) - 4\frac{5}{2} = 390$$

因此密度

$$d = 2\frac{m}{n(n-1)} = 2\frac{390}{(100)(99)} = 0.0788$$

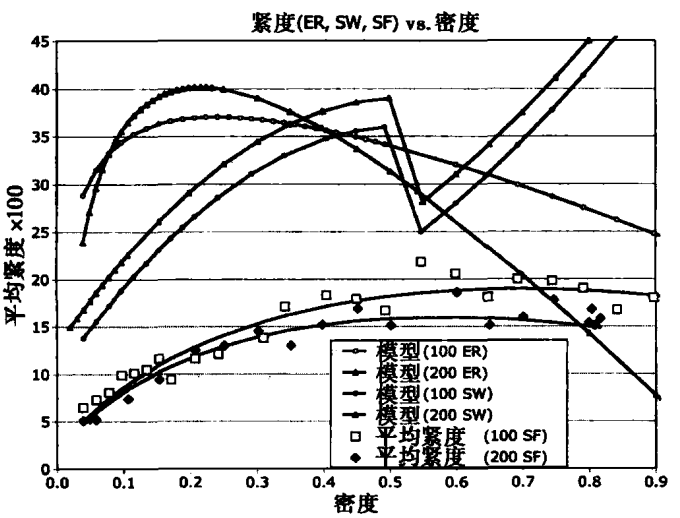


图6-7 随机、小世界和无标度网络的平均紧度与密度，显示无标度网络平均具有较弱的中间节点

平均度为 $\lambda = 7.88$，因此 hub 的期望大小近似于 $\text{hub\_degree} = 18\log_2((25)(0.0788)) = 17.6$。代入到模型中，我们就得到

$$r = \frac{\log_2(n - \text{hub\_degree})}{\log_2(\lambda)} = \frac{\log_2(100 - 17.6)}{\log_2(7.88)} = 2.14$$

$$z = \lambda^r = 7.88^{2.14} = 82.4$$

$$100(\text{closeness}) = \text{density}\, z + 1 = (0.0788)(82.4) + 1 = 7.43$$

通过图 6-7 中的仿真得到平均紧度为 8.07。因此，这种近似是中等精度的。

从该模型中可以归纳总结出以下几条：（1）平均紧度平滑地增加到一个峰值，然后随着更少的节点脱离与主导 hub 的直接连接，会平滑地下降到 50% 密度之下；（2）无标度网络的平均紧度远远低于同等的随机和小世界网络。一般来讲，无标度网络中节点的平均紧度是其他类网络节点的一半。平均起来，在无标度网络中有非常少的、不那么强大的中间节点。从某种角度来讲，hub 是唯一的中间节点，中间节点的有用性要超过所有其他节点。

平均地讲，无标度网络具有较弱的中间节点，但是 hub 要比在小世界和随机网络中可能存在的最强的中间节点要强得多。考虑表 6-1 中通过随机、小世界和无标度网络的最强中间节点，每个节点的路径数。根据网络密度，随机网络中每节点具有 0 ~ 10 条路径穿过最近的节点。例如，如果 $n = 100$，那么每节点 5 条路径意味着有 500 条路径穿过最近的节点。

对于这里研究的仿真网络来说，小世界网络的中间节点是同等随机网络情况下影响力的两倍；在同等密度下的无标度网络的中间节点是小世界网络的中间节点影响力的两倍，而是同等随机网络的 4 倍。在其峰值密度下，规模为 $n = 200$ 的无标度网络有一个带有 7600 条路径连接节点对穿过它的中间节点。

表 6-1 每个节点通过最近的节点的路径数目

网络	路径数 n
随机	0 ~ 10
小世界	0 ~ 20
无标度	0 ~ 38

6.2.5 无标度网络聚类系数

在第 4 章中，我们演示了随机网络的聚类系数随着密度线性地增加：$CC \sim O(\text{density})$。在上一章中，我们演示了小世界网络的聚类系数随着熵的增加而减少，以重联概率表示为 $CC(\text{WS}) = A - B\log_2^2(100(\text{概率}))$，因为随机性不利于聚类。换句话来讲，聚类就是一种结构，结构是一种负熵。但是对数平方表示法不能完全表达出小世界中聚类系数和密度之间的关系。

图 6-3 表明无标度网络中聚类系数与规模之间的线性关系，类似于随机网络中的行为。但是要记住图 6-3 是从变化的网络规模而导致密度的变化的数据中推导出的。因此，它本身并不完全隔离密度效应。实际上，Albert 和 Barabasi 发现在 $\lambda = 4$ 跳的无标度网络中的聚类系数按照 $CC \sim n^{-0.75}$ 与规模成反比（Albert，2002）。

在本节中，我们进行了一项实验，保持网络规模相对不变（近似地 $n = 200$）并且改变密度，同时测量聚类系数和 hub 度。我们通过仿真和曲线拟合显示无标度网络的聚类系数与随机网络的很相似：$CC = O(\text{density})$。但是之所以如此令读者感到惊讶。

近似地设置 $n = 200$ 并调整链路数产生相同规模但是不同密度的网络。特定密度的无标度网络是通过运行 BA 生成过程对于特定的 Δm 创建的。回顾无标度网络的密度按照 $2(\Delta m/n)$ 改变。这样，通过更改 Δm 我们就可以将密度调整为某值。

图 6-8 总结了小世界和无标度网络随着密度更改时测量聚类系数和 hub 度的结果。在图 6-8

中，小世界重联概率接近4%，但是更改有些是因为 Watts-Strogatz（WS）生成过程的随机特性引起的。聚类系数 CC 放大了 100 倍以便于演示。

曲线拟合确定在无标度网络中聚类系数慢慢地以密度的线性函数形式增加，但是在同等的小世界网络中增加是非线性的而且会很快。因此，我们对无标度与小世界网络会得到截然不同的结果：

无标度网络 CC:100(CC(scale-free)) ~ O(density)，1% ≤ density ≤ 10% 。这可以替换成 100(CC(scale-free)) ~ $O(\Delta m)$，因为，density = $2(\Delta m/n)$；$n \gg 1$ 。

小世界网络：CC:100(CC(small-world)) = $A - B\exp(C\ \text{density})$；1% ≤ density ≤ 10% ，这里对于 $n = 200$，$A = 60$，$B = 158.5$，$C = -100$ 。

对于小世界网络的密度和聚类系数之间的关系截然不同，因为密度的稍稍增加会导致聚类系数很快上升。它是非线性的而且要比同等无标度网络中聚类系数上升得更快。为什么？

在两类网络中，结构随着密度的增加而增加。至少某些这种结构会进入更大的聚类。但是注意图 6-8 中聚类在小世界网络中比起在无标度网络中要放大很多倍——至少在研究的密度范围内。这就是说在小世界网络中较大百分比的结构进入聚类，而在无标度网络中更大百分比的结构进入 hub 度。这一点值得重视：

结构迁移 随着密度增加，小世界网络迁移进入高度聚类网络，而无标度网络迁移进入高度 hub 网络。这是底层 k - 规则网络的高聚类系数的一个直接结果。

随着密度增加，无标度网络的 hub 度也会增加。高度 hub 倾向于降低聚类系数，因为密度线性地增加，而聚类系数必须以二次方形式增加：Degree(degree - 1)/2 。

例如，在图 6-8 中，小世界的聚类系数为 $CC = 0.60 + 1.585\exp(-100\text{density}) = 0.58$（当密度为 5% 时），同等无标度网络的聚类系数是 0.14，接近于同等小世界网络的 25% 。另外一方面，同等小世界和无标度网络的 hub 度分别为 12 和 52。无标度网络 hub 是小世界网络 hub 的 4 倍。简而言之，无标度网络具有大的 hub 和小的聚类；小世界网络具有大的聚类和小的 hub。无标度网络结构被授予高度 hub，而小世界结构被授予高度聚类。

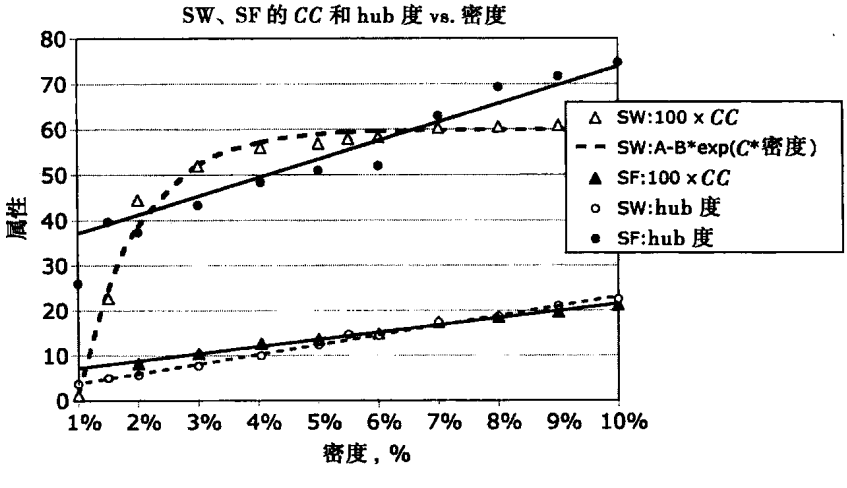


图 6-8 小世界网络和无标度网络的聚类系数与密度。对于小的密度值，小的重联概率，并且 n ~ 200，重联概率 ~ 4%（小世界）

6.3 无标度网络中的导航

有关互联网的无标度特性已经发表了很多论文资料（Huberman，1999；Kim，2002；Kleinberg，

2000a, 2000b)。众所周知，它有非常高度的 hub，并且显示成小世界效应。因此，在使用 hub 时信息应该很快地从一个节点流向另外一个节点。事实上，从一个 hub 跳到另外一个 hub 是导航通过无标度网络的一条快速路径，因为平均路径长度随着密度和 hub 度而降低。特别是，hub 度与路径长度之间的关系在使用如下所述的*最大度搜索算法*时，导致了在无标度网络中的快速导航(Paxson, 1997; Adamic, 2001, 2002)。

回顾最大度导航算法，搜索找到一条从 u_0 到 u_k 的长度为 k 跳的路径，通过以度值降序的方式访问邻接点——从最大度到最小度。在前一章中研究的*最大度导航算法*表明是无标度网络中导航的最快的方法。在本节中，我们研究密度和 hub 度对导航速度的影响，并显示 hub 度越高，导航时间就会越快。

在这种情况下的平均路径长度大于前面研究的平均路径长度，因为从节点 u_0 开始搜索到节点 u_k 与查找节点 u_0 和节点 u_k 之间的最短距离不同。网络导航缺少有关网络拓扑的全局信息，因此每一跳必须仅依赖于本地信息。对于无标度网络来说，这就意味着使用相邻节点的度来选择接下来访问的节点，如从节点到节点的搜索跳跃。

最大度导航算法简单。从节点 u_0 开始，跳到所有相邻节点中具有最高度的邻接节点 u_1，然后从节点 u_1 跳到与之相邻的具有最大度的 u_2，依次类推，直到到达目的地节点为止。当然算法要避免循环回到先前访问过的节点，当遇到死节点时就必须加以备份。在最差的场景下，最大度搜索遍历整个网络。我们假定网络是强连通的，因此每个节点都是从每一个其他节点最终可达的。

6.3.1 最大度导航与密度对应关系

最大度导航的效率如何？网络密度对于在这种网络中花费的导航时间影响如何？这些问题在对等网络设计中具有实际的应用。例如，文件共享对等网络 Gnutella 形成了无标度网络，由此 hub 自发地出现在高带宽服务器周围。因此当在 Gnutella 上搜索音乐文件时，就会搜索高带宽服务器等，直到定位需要的音乐文件为止。这种方法虽然与最大度搜索相类似但不等同，在那里高带宽服务器也是具有最高的连接数量。

图 6-9a 显示在无标度网络中所有可能节点对的平均路径长度，通过调整 Δm 从 1 到 10 来改变密度。这近似等同于当 $n = 200$ 时从 1% 到 10% 密度的更改。图 6-9a 中的点画线显示了对经验数据的接近完美的幂律分布拟合：

$$\text{avg\_hops(density)} = \frac{A}{\text{density} + B};\ 1\% \leqslant \text{density} \leqslant 10\%$$

这里通过曲线拟合得到 $A = 0.45$、$B = 0.0092$。注意这种幂律分布的指数为 1.0。在这种情况下，导航时间和密度之间成反比关系。路径的平均长度——使用最大度导航——为 $O(1/\text{density})$，因此导航通过该网络的期望时间随着密度的增加反而减少。将此种情况与平均最短路径长度比较，下降要远远慢于 $O(1/\text{density})$ [⊖]。

我们得出结论，与平均最短路径相比而言，增加无标度网络密度将会更快地减少导航时间。添加链路将缩减传送时间。但是，平均最短路径要远远小于最大度路径长度的大小，因为最短路径算法使用网络拓扑的全局知识。例如，在互联网中的分组路由使用 OSPF（最短路径优先）算法以便取得比本地最大度导航所取得的更短的平均路径长度。

⊖ 按照幂律分布，平均路径长度接近于 $O(1/\text{density}^{1/3})$。

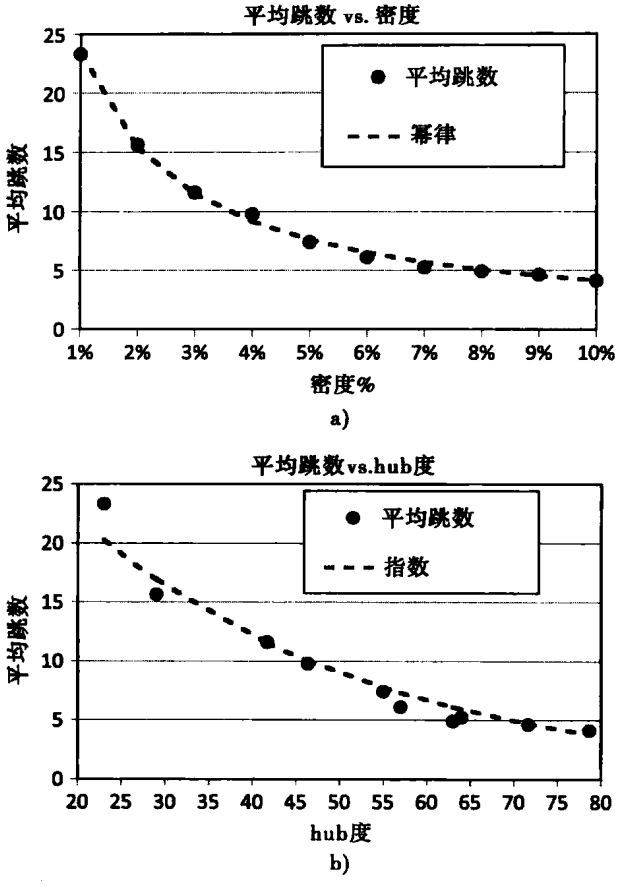


图 6-9 在规模为 $n = 200$ 的无标度网络中的平均路径长度，使用最大度导航算法：a）导航路径长度与密度对比——调整 Δm 以便得到密度；b）导航路径长度与 hub 的对比——hub 度随着密度增加而增加。路径长度使用跳来度量，网络 hub 定义成具有最高度的节点

6.3.2 最大度导航与 hub 度的对应关系

从前述的分析中我们可以预测到，随着 hub 度的提高，导航时间也会有所提高。实际上，这就是实际所发生的情况。图 6-9b 显示导航时间（平均跳数）和 hub 度之间的关系。网络的最大的 hub 度越高，信息通过无标度网络的速度就越快。但是，该关系拟合指数曲线而非幂律分布：

$$\text{avg\_hops}(\text{hub\_degree}) = A\exp(-B(\text{hub\_degree} - C)); 20 \leqslant \text{hub\_degree} \leqslant 80$$

这里对于 $n = 200$，$1\% \leqslant \text{density} \leqslant 10\%$，$A = 20.25$，$B = 0.03$，$C = 23$。hub 度随着密度呈对数增加，因此该结果与网络密度相关。hub 度对导航时间具有很大的影响。hub 度越高，无标度网络导航就越快。直观来讲，这很有道理，因为 hub 倾向于直接连接到较大百分比的节点上。它们提供许多直接到每个邻居的远距离路径，需要较少的短距离“跳”到达最终目的地。在这种情况下直觉被证明是正确的，但是总是需要用实验结果来证实学术上的猜测才能放心。

这种结果对于设计如 Gnutella 及其他对等网络很重要，因为它具有高连通的 hub，假定在 hub 处的服务器可以跟得上带宽需求。无标度网络导航本质上要比随机或小世界网络更加有效。它们既具有小世界效应又具有低聚类。在某种意义上，无标度网络要比小世界网络实现得更好，由此得出结论：无标度网络“比小世界更小”！

这里最后一个观察是按顺序。在大多数情况下，无标度网络的 hub 也是具有最高紧度特性的节点。hub 也是最强的中间节点——并且因为它们要比其他节点捕获更多的路径，它们需要极高

的带宽。因此，最大紧度搜索等同于最大度搜索。但是计算节点的度要比计算紧度值更加容易。因此，我们不必麻烦比较最大紧度和最大度之间的不同。

6.3.3 在无标度 Pointville 网络中的弱联系

在前一章中，我们使用近似方程求解估计由 $n = 129$ 个市民组成的 Pointville 的隔离度或弱联系长度的实际问题，每个市民平均认识 λ 个其他市民。在本章中，我们假定 Pointville 构成一个无标度网络，并估计它的直径和平均路径长度。从本章中我们学习到，无标度 Pointville 应该具有强的弱联系（小世界效应），从而导致小的直径和低的平均路径长度。

回顾，$n = 129$，$m = 1548$，而 $\lambda = 24$，随机 Pointville 的直径就是2.36跳，而对于重联概率为 $p = 5\%$ 的小世界直径为3.5跳。这些网络的密度近似于

$$d = 2\frac{1548}{(129)(128)} = 18.75\%$$

同等无标度网络的平均路径长度和直径是多少？

一个任意密度的无标度网络由输入参数 n 和 Δm 定义。我们已知 $n = 129$，但是 Δm 等于多少？求解密度方程，我们就得到：

$$d \sim 2(\Delta m/n), \text{对于 } n \gg 1$$

重新安排项产生 $\Delta m = d(n/2) = 0.1875(\frac{129}{2}) = 12$。将这些数字插入到程序 Network. jar 中并平均进行三次实验，生成直径、半径和平均路径长度的估计。根据经验：

$$\text{Diameter(无标度 Pointville)} = 3 \text{ 跳}$$

$$\text{Radius(无标度 Pointville)} = 2 \text{ 跳}$$

$$\text{avg\_ path\_length（无标度 Pointville）} = 1.85 \text{ 跳}$$

从前一节中导出的近似方程中，我们可以得出：

$$10(\text{avg\_path\_length(Pointville)}) = 14\frac{\log(n)}{\log(n) + \log(2\text{density})}$$

$$= 14\frac{\log(129)}{\log(129) + \log(2(0.1875))}$$

$$= 14\frac{2.11}{2.11 - 0.426} = \frac{29.54}{1.684} = 17.54 \text{ 跳}$$

$$\text{avg\_ path\_length(Pointville)} = 1.75 \text{ 跳}$$

这种结果证实了我们已经学习到的有关无标度网络的知识——与随机和小世界网络相比它们相对“小”。事实上，无标度 Pointville 比起小世界 Pointville 来说是一个更小的世界；它需要平均1.85跳才能遍历无标度 Pointville，平均2.17跳遍历同等的小世界 Pointville。这是由于在无标度网络中要比同等小世界网络中存在大量远距离链路。无标度网络具有更强的弱联系！

从社会网络理论角度来讲，无标度网络具有少量节点连接到非常大百分比的所有其他节点上，并且很多节点非常稀疏地连接到 hub 上。无标度网络显示出非常大的中心性——无标度 Pointville 的半径也是非常小的。大多数节点非常靠近所有其他节点。例如，无标度 Pointville 中所有其他节点的三分之一节点到中心只有两跳，而在小世界 Pointville 中所有节点中仅有10%节点到中心是10跳。

无标度 Pointville 网络甚至超出了星形网络——一个 hub 节点按照平均路径长度连接到所有其他节点。不管星形网络增长到多大，每一个节点离其他每个节点不超过2跳。当然，如果 Pointville 是一个 $n = 129$ 的星形网络，它的直径将是2，而它的平均路径长度将接近于1.95跳——将比无标度 Pointville 大0.1跳。

6.4 分析

按照网络拓扑范围，无标度网络介于小世界和随机网络之间。除了聚类系数之外，无标度网络类要比其他三类更加平衡（参见图6-10）。熵、平均路径长度和hub度相对高，但是聚类系数较低。这些属性主要关系到无标度网络是如何实现的。

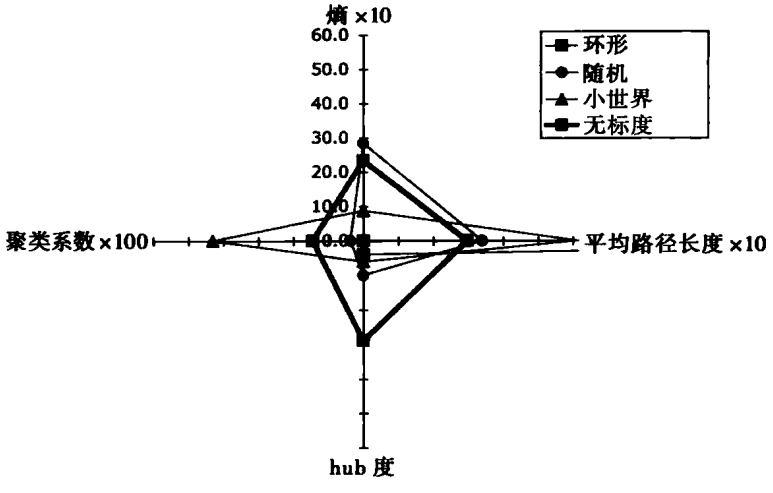


图6-10 环形、随机、WS小世界和BA无标度网络属性的Kiviat图比较

6.4.1 熵

熵 $I(\text{scale-free}) \sim O(\log_2(\Delta m))$ 以偏好连接参数 Δm 的函数增长，但是 $\Delta m = \text{density}(n/2)$，因此 $I(\text{scale-free}) \sim O(\log_2(n/2))$。将此与随机和小世界网络的熵相比较。小世界网络的熵，$I(\text{small world}) \sim O(\log_2(100p))$，这里 p 是重联概率。因此，从熵的方面来讲无标度网络更像小世界网络而不是随机网络。但是当然，小世界网络的熵放大了，因此随着密度增加相似性减少。

作为对比，随机网络的熵会随着密度增加而增加，到达峰值，然后随着密度接近100%而降低到零。无标度网络要比随机网络更结构化？极高密度随机网络不是纯的随机网络，因为随着密度接近100%变成规则的（完全图）。当它们也增加密度时，相似的随机损失困扰着无标度网络。随着密度增加，无标度和随机网络两者都呈现结构化。这就意味着随着密度到达100%，熵降低到零。例如，75%密度的无标度和随机网络的熵在 $n = 75$ 时分别等于4.4和3.8比特。随着密度接近100%，这种差距很小——在90%密度时，无标度熵降低到3.8比特，随机熵降低到3.2比特。在99%密度时，两类网络的熵小于1比特，两类网络在100%密度时都降低为零。

即使完全网络和稠密无标度网络的度序列分布行为很相似，它们之间也存在结构性差异。随着网络变得更加稠密，因此就会更像完全网络，它的度序列分布呈现尖峰——所有节点接近同一度。在到达100%密度之前，随着接近两种状态之一，度序列分布变得很窄：它们既可以是（1）高连通的（高度）以至于只有一步就到达所有其他节点；或（2）离开所有其他节点两步。换句话来讲，半径序列分布具有两个频率：$R(1)$ ——部分离其他节点为1跳的节点，$R(2)$ ——部分到所有其他节点距离为2跳的节点。这种现象发生在非常稠密的无标度网络中，如图6-11所示。

在无标度网络中随着密度接近100%，熵呈指数级下降。之所以发生这种现象，是因为度序列分布崩溃的结果。所有节点都最大限度地连接起来，以至于所有节点彼此之间距离是1或2跳。实际上，在熵减少的同时网络中心的节点数增加到100%。相似地，离所有节点距离为2跳

（直径）的节点数量减少。对于非常高密度来讲，从无标度网络向完全网络的转变呈指数级增长，参见图6-11中的点画线。

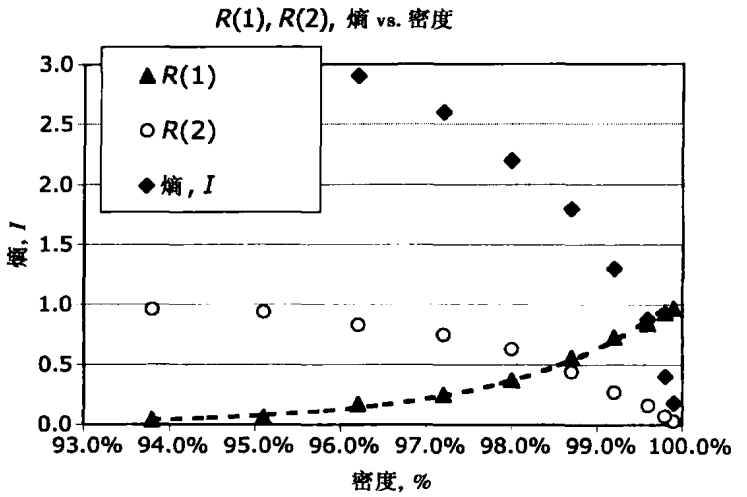


图6-11 增加半径长度1，减少半径长度2，与无标度网络的非常高密度对比。$R(1)$的上升几乎与指数函数完全拟合，图中显示为虚线

6.4.2 路径长度和通信

固定尺寸的稀疏无标度网络（对于固定的n）的平均路径长度按照线性关系 avg_path_length $= A - B$ hub_degree 随着 hub 度（密度）的增加而降低。

换句话来讲，对于固定的n，平均路径长度按照$O = \left(\frac{\log(n)}{\log(n) + \log(\text{density})}\right)$或者按照 log (density) 的反函数减少。但是图6-3明显与图6-5和这里的结论概括相矛盾。为什么？图6-3显示同时更改规模n和密度d，而图6-5仅更改密度。事实上，图6-5显示$n = 200$的关系。当删除网络规模n的效应时，这就仅留下密度作为平均路径长度的独立变量。因此，有两个不同的平均路径长度的近似：avg_path_length(n)和 avg_path_ length(density)。

路径长度是网络理论许多应用中的一条重要属性。或许最明显的应用在于分布式计算机系统的设计。例如，考虑构建由n个处理器和m条通信链路构建的网络多处理器系统。假定我们将处理器按照 BA 算法相互连接起来，获得低费用和高性能的“最好的”处理器和链路平衡是什么？换句话来讲，优化性能需要的最小的链路数是多少？

为了便于说明，假设$n = 16$个处理器，以平均路径长度和链路数m定义“成本效益”如下：

$$\text{Effectiveness} = \frac{1 - \text{avg\_path\_length(density)}}{m}$$

$$\text{Density} = 2\frac{m}{n(n-1)}$$

在该公式中，减少 avg_path_length 或m，或同时减少两者都会提高效益。但是减少m会增加 avg_ path_ length，因为：

$$\text{avg\_path\_length}(m) = \frac{A\log(n)}{\log(n) + \log(C\ \text{density})}$$

对于$n = 16$个处理器来讲，这里$A = 0.75$和$C = 1.5$。将密度替换成m的函数就得到：

$$\text{avg\_path\_length}(m) = \frac{A\log(n)}{\log(n) + \log(C) + \log\left(\frac{2m}{n(n-1)}\right)}$$

$$= \frac{A\log(n)}{\log(2mC) - \log(n-1)}$$

$$= \frac{0.75\log(16)}{\log(3m) - \log(15)}$$

如此一来：

$$\text{Effectiveness} = \frac{\dfrac{1 - 0.75\log(16)}{\log(3m) - \log(15)}}{m}$$

可以通过增加分子、减少分母或两者同时进行来最大化效益。假定固定 n，找到一个最大化效益的“平衡值” m：$\max \{\text{effectiveness}\}$。

求解该问题的策略是：

1. 在包括最大值 effectiveness（m^*）的一段范围值内画出效益与 m 之间的关系图。

2. 找到产生一个近似有 m^* 条链路和 $n = 16$ 个节点的无标度网络的 $\Delta m \sim (m/n)$。

BA 生成过程不能产生任意密度的无标度网络，但是相反能产生对应于 Δm 整数值的一系列范围的密度。整数值 $\Delta m = [m/n]$ 或许会产生刚好 m 条链路的无标度网络。在该例子中，$[m/n]$ 与 $[m^*/n]$ 之间的差距很小。例如当 $A = 0.75$，$B = 1.5$，$n = 16$ 个处理器时，我们得到

$$m^* = 84 \text{ 条链路}$$

$$\Delta m = 7$$

$$m = (\Delta m)n - \Delta m\frac{\Delta m + 1}{2} = 84 \text{ 链路}$$

$$\text{密度} = 2\frac{m}{n(n-1)} = 2\frac{84}{(16)(15)} = 70\%$$

$$\lambda = n\text{density} = (16)(70\%) = 11 \text{ 条链路}$$

根据这里建议的简化效益度量，当密度很高时，16 个处理器的无标度网络是最优的。事实上，这种无标度网络几乎可用 hub 度为 15 的、有近似 8 条链路的最小度节点未完成。验证该结果也适用于其他的 n 值，这留给读者作为练习。

6.4.3 聚类系数

无标度和随机网络具有相对较低的聚类系数，会随着密度的增加而线性地增加：$O(\text{density})$。将此与由 WS 生成过程（密度 = 4%）构建的小世界网络中的聚类相比（参见图 6-8）。表面上看来聚类与 hub 度相关。它们两个都随密度的增加而增加，但是 hub 度增加得更快。

围绕无标度网络节点的本地聚类的确与节点的度相关，但是以相反的方式。度越低，聚类就越高。直觉感受 hub 是高聚类系数邻居的中心，但是事实并非如此。图 6-12 显示这里研究的每一类网络的本地聚类系数对节点度的影响。本地聚类系数是与每个单独节点相关的聚类系数。网络聚类系数则是所有节点的平均值。

在图 6-12 所示的无标度网络中，曲线显示成一条受节点的聚类系数约束的实线，但是在其他类网络中则不受节点的聚类系数约束。在无标度网络中，本地聚类系数和节点的度相关，在其他两类网络中这种相关就不存在。

图 6-12a 演示了本地聚类系数是如何均匀地分散到随机网络所有度的节点上。显然，在随机网络中聚类系数与节点的度不相关，但是这要留作读者练习的研究项目。在图 6-12a 中，我们推测本地聚类系数与 hub 度之间的关系是随机的[⊖]。

⊖ 对于随机网络 $CC \sim (\lambda/n) = \text{density}$（Albert，2002）。图 6-12a 证实了这种理论估计：$CC \sim \frac{8}{100} = 0.08$。

图6-12b演示了在小世界网络中本地聚类系数要比其他两类网络更高——正如预期的那样。本地聚类系数与hub度之间的关系图位于上面，下面的实线是通过以下关系获得的：

$$CC(\text{scale-free node}) \sim \frac{(1+m)/n}{\text{degree}(\text{scale-free node}) + 1 + (m/n)}$$

此外，本地聚类系数值倾向于集中于具有中等度的节点。但是这种推测也留给读者作为练习进一步探索。如果在图6-12a中的画图是随机的，那么在图6-12b中的画图是非随机的。

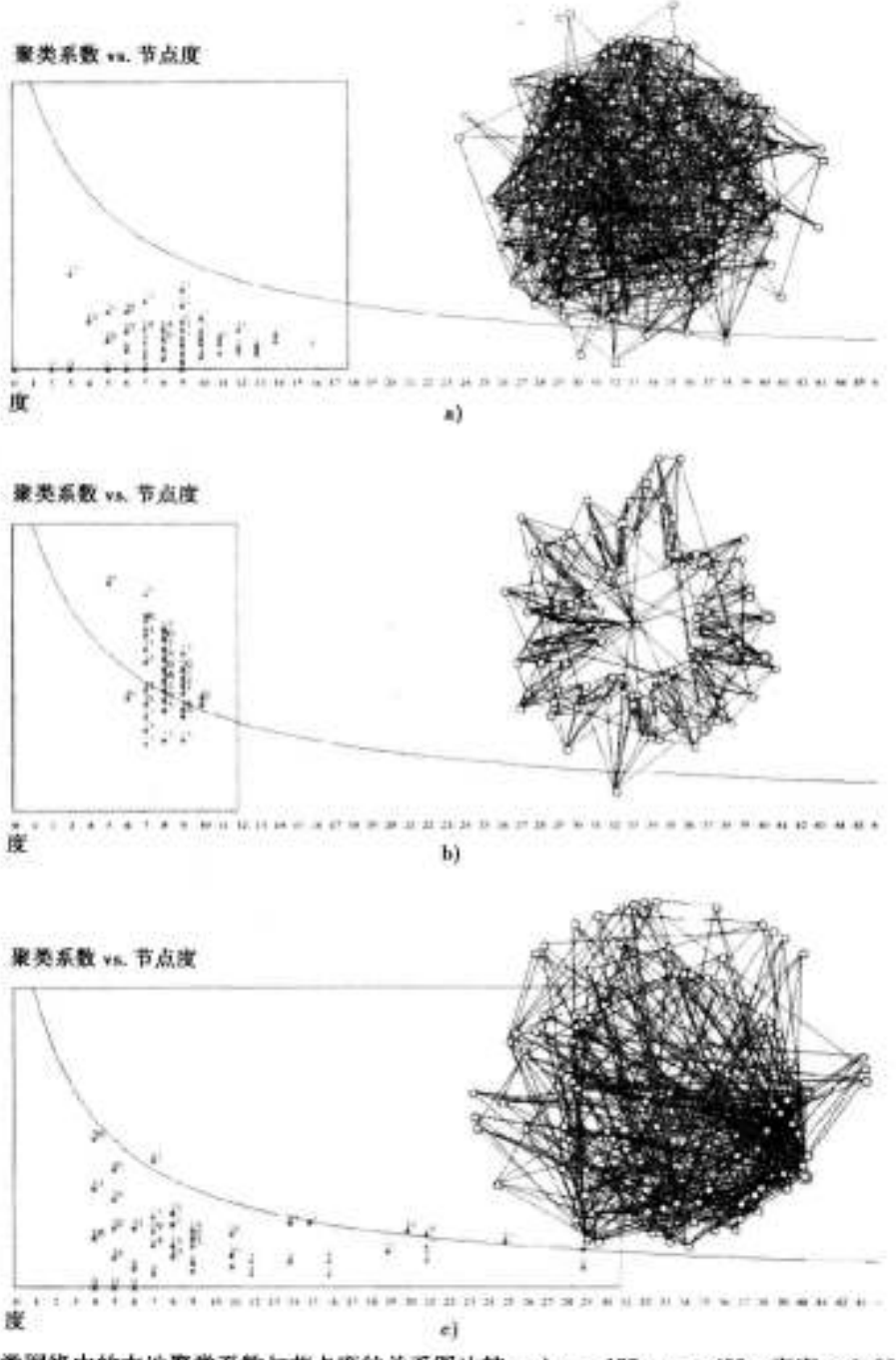


图6-12 每类网络中的本地聚类系数与节点度的关系图比较：a) $n = 100$，$m = 400$，密度 = 8.08%，$\lambda = 8$ 的随机网络；b) $n = 100$，$m = 400$，密度 = 8.08%，重联概率 = 10% 的小世界网络；c) $n = 100$，$\Delta m = 4$，密度 = 7.87% 的无标度网络。每个网络都是围绕中心节点画图的（半径）

图6-12c支持以前做出的推测——在无标度网络中节点的度越高，其聚类系数就越低。的确，本地聚类系数对于低度节点要比高度节点的更高。令人惊讶的是，高连通节点具有较低的聚类系数。但是这种效应会随着密度的提高而消失，并且无标度网络变得更像完全网络。对于高密度（>5%～10%），这种相关性失效并且聚类系数会更像随机网络的。

在图6-12c中所示的无标度网络的上限是从聚类系数和度之间的反向关系获得的：

$$CC(\text{scale-free node}) < \frac{\Delta m}{\text{degree}(\text{scale-free node})};\ \Delta m \ll n$$

这种上限对于低网络密度（Δm）来说非常准确，因为无标度网络的熵也会随着密度的降低而减少。随着密度的提高，熵也会提高，无标度网络变得更加随机。随着熵的提高，图6-12c具有图6-12a的属性。

稀疏无标度网络包含具有低聚类系数的hub，其他节点具有相对高的聚类系数。该结论对于社会网络来说具有重要的应用，因为社会网络倾向于稀疏的。具有最高连通性的节点显示成连接了最少数邻居节点的节点。这就意味着最受欢迎的演员（hub）可能不属于一组好朋友？如果果真如此，这似乎与社会影响力大部分赋给了hub的理念相矛盾。为什么hub具有低的聚类系数呢？

回顾聚类系数就是从零到degree(degree－1)/2的部分范围。也就是，$CC \sim O(1/\text{degree}^2)$。随着节点度的增加，必须相互连接的邻居节点数随着度的平方增加。对于高度节点，有时这很难实现。在图6-12中所示的结果大部分（但是不是全部）可以由聚类系数来定义。结果表明，本地聚类的上限与度成反比，而非degree^2成反比。很显然，聚类的发生并不像$O(\text{degree}^2)$那么快。

6.4.4 hub度

无标度网络通过从随机连通性移动到像hub的连通性来展示结构。作为回报，它们牺牲了聚类和熵。直观来讲，熵在无标度网络中变得像hub似的拓扑，在小世界网络中变得聚类化，而在随机网络中则变得无序。这就是为什么无标度网络类被称为“带有hub的类”，以及小世界网络类被称为“带有聚类的类”的原因。

无标度网络将很大部分网络节点连接到很少一部分节点上。这种简单体系结构影响从平均路径长度、快速导航到本地聚类等几乎所有类的独特特性。hub度随着密度的增加而增加（参见图6-4）。直观来讲，小世界网络是一种带有hub度约束的无标度网络，而无标度网络却没有这种约束。如果最大hub度受限制，无标度网络的属性会有什么改变？这个问题留给后面章节继续研究。

练习

6.1 证明泊松分布函数不是无标度的，因此随机网络不是无标度网络。

6.2 利用密度推导出无标度网络的熵表达式。

6.3 在包含$n = 100$个节点，$m = 1000$条链路，hub度为50的无标度网络中，平均路径长度是多少？应该使用哪个公式计算？

6.4 对非常稀疏的随机网络（$n = 200$，密度2%～10%）进行实际研究，并推导出一个平均路径长度和hub度之间的近似关系。平均路径长度会随着hub度的增加而减少吗？正如无标度网络中那样的吗？（参见图6-13。）

6.5 优化$n = 24$个处理器的无标度网络的效益时，m等于多少？Δm等于多少产生对最优m的最佳近似？

6.6 $n = 500$和$\Delta m = 2$的无标度网络的近似聚类系数等于多少？

6.7 对 $n = 200$ 和密度 =40% 的无标度网络中的聚类系数和节点度之间的相关性进行实际研究。有关相关性我们会推导出什么结果?

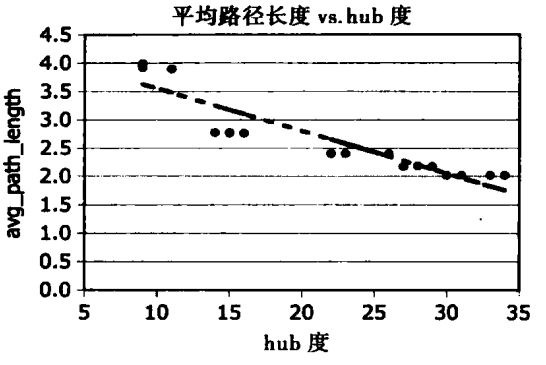


图 6-13 平均路径长度与 hub 度对比

6.8 具有 2% 的密度并且 $n = 500$ 的无标度网络中度为 9 的节点的近似本地聚类系数是多少? 将理论估计值和经验导出值进行比较。

6.9 对于最大度搜索导航, $n = 500$ 和 $\Delta m = 2$ 的无标度网络的搜索路径长度是多少?

6.10 $n = 200$ 和 $\Delta m = 2$ 的无标度网络的最大的 hub 的期望最大度是多少? 随着 Δm 从 2 增加到 10 这种最大值如何更改?

6.11 在练习 6.10 中网络的平均紧度是多少?

6.12 $n = 500$ 和 $\Delta m = 5$ 的无标度网络的平均紧度是多少? 为 $n = 200$ 设计的经验模型能够精确地估计该网络的平均紧度吗?

第7章
Network Science: Theory and Applications

涌　现

在一个静态网络中，节点、链路和映射函数的属性都不会随着时间而变化。在一个动态网络中，节点和链路的数量、映射函数的形状及一些其他属性可能会随着时间而变化。动态网络是一个时变网络。通过 BA 增长算法生成的无标度网络是一个简单的动态网络的例子，它是从少数初始节点和链路演化而来的。

时变更改导致网络中的结构化的重组——在某些学科中称为进化——在本书中称为涌现。通过涌现形成的网络称为涌现网络，它的形成是先从一些预定义的初始状态（即随机网络），然后通过一系列小的变化逐渐变成最终状态（即无标度网络）。这些小的改变是由微规则所定义的，它从内部、外部对网络产生影响。外力一般是来源于环境力量，而内力则经常来源于遗传力量。

涌现——通过不断重复的小的变化引起的网络特性的转变，是本章中将要讨论的一个过程，并作为随后几章内容的一个序曲。我们将演示通过涌现过程可以构造任意结构的网络，具有几乎任何想要的特性，如高度的 hub、高聚类系数或特定的度序列。事实上，我们将会演示非常小的微规则在将任意网络重塑成带有特定属性的“设计者的网络”时是多么有效！

通过涌现我们可以看到两个基本的原则：（1）通过不断重复应用非常简单的规则构造任意结构是可能的；（2）现实世界的网络结构，不管是生物、物理还是社会网络，都可以解释成简单涌现的结果。例如，美国电信网络的结构是 1996 年电信法案的直接结果，这个法案促成了高度集中的、交换 hub 的形成，现在被称做“运营商数据交换中心”。电信的无标度结构是对等的直接结果，对等是法律所倡导的。这个简单的规则导致了美国国家网络的大规模重组。

简单微规则的重复利用也是一个与实际系统中的优化相类似的过程。网络通过优化自身的一些特性，比如最小化路径长度、最大流量或最小代价，频繁、快速地对外部条件做出响应。通过对外部刺激做出响应，系统经常改变自己的结构或是优化自己的一个或多个属性来加以适应。我们会通过一个简单的涌现过程来最小化电路中连线的总长度来说明这一原则。

我们研究涌现网络的很多简单应用，并说明如下：

1. 涌现在不同的研究领域有着很多不同的定义。我们将网络涌现定义成一个动态网络过程，由此宏观尺度属性如度序列的涌现是不断地应用微观尺度规则（如简单的链路交换）的结果。

2. 一个动态网络如果它是收敛的话，经历涌现后在有限的时间就会到达最终状态——否则它就是发散的。发散的过程会导致网络出现静态或振动的宏观结构。涌现网络的稳定性将在下一章中讨论。

3. 开环涌现（遗传涌现）通过反复地应用微规则不受外部或者环境影响地重构网络。当一个动态网络由外部或者环境影响重构时，过程定义了一个反馈循环涌现（环境涌现）。反馈循环涌现是具有自适应性的，而开环涌现是非适应性的并且仅依赖于内部微规则。

4. 涌现建模微生物、物理、数学和计算科学中的系统行为。特别地，我们会将涌现理论应用到在印刷电路板上最小化地构造连线总长度的问题上。这个问题与自然界和人造世界中遇到的很多优化问题相似。

5. hub 涌现是一个开环涌现过程，它通过创建与无标度网络中相似的 hub 来重构任意动态网络。然而，由此产生的网络不是无标度网络，即使最大的 hub 尺寸在时间 t 按照 hub_degree(G) = A + (sqrt(t)/2) 增加到 O(sqrt(t))。熵按 $I(G) = A + B(1 - \exp(-Ct)) - D$ hub_degree (t) 增长然后减少，这里 A、B、C、D 是通过实验确定的参数。

6. 聚类涌现是一种开环涌现过程，它是按照 $CC(G) = A + B(1 - \exp(-Ct))$ 来增加动态网络的聚类系数的。但是，结果网络不是小世界网络，因为它的平均路径长度可能相对较大。

7. 度序列涌现是一种开环涌现过程，它重构网络以便于它的初始度序列 $g(0)$ 可以与由 g(final) 定义的任意可实现的序列相匹配。如果这种网络是可以实现的，Molloy-Reed 过程会生成一个具有预定义度序列 g 的网络。Erdos-Gallai 条件是网络可实现的必要条件，但不是充分条件。

8. 具有度序列 g (final) 的网络 G 使用 Mihail 生成过程更容易实现。在连接到较低度的节点之前，通过将最高度节点相互连接，Mihail 等人建议一种满足 Erdos-Gallai 条件的涌现过程 (Mihail, 2002)。如此一来，就可能产生一个由 g (final) 定义的拓扑并对 g (final) 做某种限制的设计者的网络。

9. 链路排列微规则将固定点变换应用到网络 G，这样链路排列不会更改 G 的度序列。如此一来，很多同态网络由链路排列变换产生，例如交换随机选择的链路对的端点。固定点链路排列被用来变换网络属性，同时保留其度序列。

10. 聚类系数排列涌现是一种反馈循环涌现过程，该过程通过重复应用链路排列变换提高 G 的聚类系数，但不更改其度序列属性。无标度或随机网络仍旧是无标度或随机网络，而聚类系数增加到接近于小世界网络的聚类系数。

11. 具有固定度序列 g 的网络 G 的总链路最小化是通过重复应用链路排列和防止总链路长度增加的反馈循环涌现的。假定具有度序列分布 $g(0)$ 和初始的总链路长度为 $L(0)$ 的网络 G，网络 G (final) 涌现按照时间与 $O(C_2^m)$ 成正比[⊖]，其中 L (final) 是可能的最短长度。这种涌现过程可以找到连接印刷电路板组件所需要的最少量的线。总的来说，涌现的链路最小化按照预定的度序列 g 找到连接节点的最佳方法，同时最小化总的链路长度。

7.1 什么是网络涌现

在动态网络中，$G(t) = [N(t), L(t), f(t): R]$ 是一个时变三元组，由一组节点 $N(t)$、一组链路 $L(t)$ 和将链路映射到了一对节点的映射函数 $f(t): L(t) > N(t) \times N(t)$ 组成。$N(t)$、$L(t)$ 和 $f(t)$ 是随着时间变化的，因此 $G(t)$ 标示成动态网络。例如，网络的形状由映射函数 $f(t)$ 来定义，它可能随着时间的推移在删除、添加和重链（从一个节点换到另一个节点）节点时变化。

假定网络 $G(0)$ 在它的初始状态，将规则集合 R 中的一个或多个微规则在每一个时间步应用到 G 上，就从 $G(0)$ 到 $G(1)$、$G(1)$ 到 $G(2)$ ……迁移，直到达到某一最终状态 $G(t_f)$ 为止，或重复之前的状态：

$$G(t+1) = R\{G(t), E(t)\}$$

这里 $E(t)$ 是作用在 G 上的外力。如果 G 到达了终态 $G(t_f)$，对于所有 $t \geqslant t_f$ 仍然保持不变，那么 G 是收敛的，否则就是发散的。$G(t_f)$ 又被称为固定点，因为 G 一旦达到 $G(t_f)$，它就永远保持不变了。当 G 是收敛的时，我们就说 $G(0)$ 收敛到 $G(t_f)$；否则，G 就是发散的。如果对于所有初始状态 s，$G(s)$ 收敛，我们就说 G 是强收敛的[⊜]。

⊖ 这表示成 $C_2^m = m((m-1)/2)$。

⊜ 从混沌理论中借用，我们可以讲强收敛网络在有限的时间里搜寻其奇异吸引子 (strange attractor)。

一个涌现网络不一定是收敛的。然而，在本章中我们研究的状态变化都是发生在有限时间里，$0 \leqslant t \leqslant t_f$，我们一般不关心更大的问题：网络是不是收敛[⊖]。动态网络可能在两种状态之间抖动，在从初态发展到终态，或者是一直变化着而无法达到它的“无限”状态。忽略这些可能性，在本章中我们在有限的时间里研究动态网络：从某个初始状态开始，在微规则 R 的有限数量的应用下发展到某个其他的状态。我们有兴趣观察或许预测一下，在微规则的有限数量的应用中这种网络的行为。

7.1.1 开环涌现

在图 7-1 中显示了本书中用到的两个涌现模型：开环和反馈循环涌现。*开环模型表示了遗传或内在的网络进化。反馈循环模型表示了适应性网络进化*，即网络适应环境和外部的反馈。在这两种情况下，我们假设存在一组微规则，代表内部消耗的能源或外部作用到网络上的力量。

从某个初始状态 $G(0)$ 开始，应用一系列微规则，致使网络进入到下一个状态，这样就产生了一个修改过的网络 $G(1)$；再次应用微规则产生网络 $G(2)$；依此类推。如果过程在某个时间点是收敛的，$G(t)$ 就不会更改，因为 G 已经到达了它的终态。如果过程是发散的，那么到新状态的迁移还会永远继续下去。

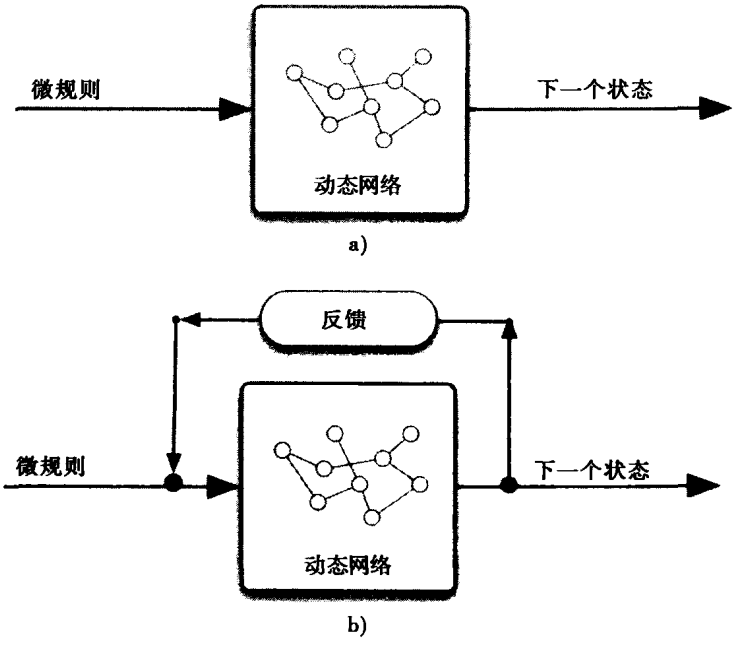


图 7-1 动态网络中的涌现：a）开环涌现，典型内在的或遗传进化；b）反馈循环涌现，一般外部或环境力量形成网络

例如，微规则可能会决定链路是如何重联的，或者对节点度设置限制。重联一个简单的链路会创建一个新的网络拓扑，表示在 $t+1$ 时的网络状态。如果在某个时间点，没有别的链路被重联，过程就是收敛的了，就涌现网络 $G(\text{final})$。否则，网络永远无法到达终态，而是经过上百万次没有终点的状态一直循环。这样的过程就是发散的。

一组微规则 R 将网络 $G(t)$ 映射到它的下一个状态，$G(t+1)$：

$$G(t+1) = R\{G(t), E(t)\}$$

这里 $E(t)$ 是在时间 t 的反馈信息集合。在图 7-1a 中只应用没有反馈的微规则。下一个状态仅由

⊖ 稳定性将在下一章中研究。

微规则来决定，通过其状态的改变来过渡到网络的下一状态。这些微规则直接反复地执行一些很简单的操作，比如增加链路的数量，增加节点的度，限制链路的长度。它们的作用效果是局部的，并不是全局的；也就是说，微规则只是在局部层次引起一些改变，比如重联一条链路会让一个节点的聚类系数或度增加。由于微规则只作用于内部的节点和链路，所以我们将开环进化归类成一种遗传涌现。

开环涌现经常会产生意想不到的结果或是无法预测的网络全局属性。这就是涌现区别于其他形式的网络变换的一个方面。从根本上来讲，涌现就是在局部通过不断应用小的变化获得全局模式的过程。在开环系统中，网络的终态和微规则之间并没有明显的关系。然而，在绝大多数情况下，在涌现显示出来后，为什么重复应用微规则会导致全局模式就很明显了。

7.1.2 反馈循环涌现

图7-1b定义了另外一种涌现的过程模型——*反馈循环涌现*，在这种情况下网络会经历状态的改变来满足某个全局目标，一般是由全局因素指定的，比如像网络连接的长度总和，最大流量，或者是节点的总度数。因为要目标是调整适应外部条件或者限制，这个模型也同时被称为*适应性涌现*或*环境涌现*。除了微规则效应外，反馈循环带来的改变通常比微规则带来的影响要大。于是，过程还是像之前一样继续应用微规则，但是它们的作用是受反馈循环信息的控制和指导的。一般来讲，反馈是对外部定义目标的性能或紧度的全局测量。

例如，假如全局环境目标是最大化网络的hub的度，并且有一个微规则是随机地重联链路。在每次应用微规则后，我们就进行下面的修正：（1）当重联链路的结果会减少节点度时就将这个微规则删除；（2）否则，就允许这个微规则继续起作用。在这个简单例子中，反馈信息是网络hub的度。无论何时微规则增加了节点的度，就让它继续工作，我们会让任何减少节点度的规则应用失效。通过这种限制会生成什么形式的网络呢？我们会在本章的下一节中详细研究hub涌现。

在反馈循环模型中，涌现还有一个目的——达到某个全局的目标——这是一个逐步增量实现的目标。因为明显的原因，我们称这样的涌现网络为*面向目标的涌现网络*。*面向目标的涌现*是通过反复地应用微规则R和“修正”反馈信息来改变动态网络全局属性的一个过程，这些属性包括直径、度序列分布、所有链路的长度总和等。

记住反馈循环涌现和开环涌现的不同——前者是适应性的，外部或环境的压力会决定网络的下一状态。这通常意味着覆盖微规则——一般以*惩罚*或*奖励*反馈机制的形式叠加在微规则上[⊖]。反馈机制一般是与一个目标或目标功能相联系。随着时间的变化，目标会变成网络的一个涌现特性。

开环涌现仅对网络的内在的或者遗传属性做出响应。它会忽略所有的外部和环境因素，只有节点和链路的局部属性，比如度、中心性、聚类系数决定着网络的下一状态。如果一个全局属性能够通过微规则的反复应用而进化，这是最好的。但是目标是局部的，不是全局的。开环涌现是非适应性的，但是也可以进化。例如，hub涌现是一个增加任何网络的hub的度的涌现过程，即使初始网络是随机的。不需要反馈循环，而需要通过简单重联的多次应用就可以生成的一个高度hub。

7.2 科学中的涌现

在很多学科中，涌现很常见，尽管所使用的术语和我们在这里使用的有点不同。实际上，各种生命形式的进化与图7-1中给出的模型之间的相似并不是巧合。我们会在工程、物理和生物学

⊖ 如何“惩罚”一个网络？直接使会导致一个不需要的更改的微规则应用无效。

中发现这些模型。网络科学提供了专门为每个学科量身定制的一种抽象表示。下面对涌现在社会和物理科学中的应用的概括只是一个开端，更多详细的模型会在本书其余部分设计出来。

7.2.1 社会科学中的涌现

在社会科学中，N 是个体的集合，L 定义成个体之间的关系。例如，N 可能是办公室中在一起工作的一群人，L 可能就是组织结构图中的线条。或者，N 是代表了可从其他人那里感染传染病的一群人，L 可能就是导致疾病传染的交互。

人群中的社会结构的涌现是一种巨大、复杂的现象。网络科学试图通过研究动态网络的形式和社会网络内部的进化变革来解释这种现象。例如，在最近的邻居间形成的链路就倾向于比远距离链路具有更高的概率。这样会造成本地的高聚类、短的平均路径长度和非随机的结构（Davidsen，2002）。

涌现不仅是网络从一个初始状态到终态的转换。在物理和生物科学中，“涌现是系统中产生了某个新的现象的概念，但该现象又不在系统开始的规范中”（Standish，2001）。这个定义涉及微规则的反复利用，导致意想不到的宏观结构。例如，网络的度序列分布是一种描绘其宏观结构的方式，而链路重联微规则可能描绘它的微观结构。在度序列分布和连接节点对的某个规则之间是没有明显关联的。不过，某些“新现象的出现”可能意想不到地从简单重联的多次应用中产生。例如，“新现象的产生”可能是从进化中出现的一个无标度度序列分布，即使“无标度结构”不在系统的初始规范中。这种“新现象”是不可预期的，因为偏好连接在局部层次工作，但是度序列分布是一个全局属性。

7.2.2 物理科学中的涌现

在物理科学中，涌现用来解释一些比如材料中的相变（气体冷却转变成液体等）或者 Ising 效应（磁性极化）的现象。例如在热力学中，涌现将物质的大规模宏观属性与它的微观状态相联系。具体来讲，将一块冰的温度（宏观属性）与它的分子（微观属性）相联系，在水到达冰点之下，每个分子会根据物理学的微规则来改变相，水的形态也会由液态变成固态（宏观属性）。在网络理论中，这就相当于将网络的分类与它的熵相关联，随机网络的熵会比同等的规则网络的要大。那么在什么条件下随机网络会变成规则网络[⊖]？

涌现的出现源于原子级别（例如，节点和链路层次）的微观行为。它是从微观规则中产生宏观的模式。通常在宏观和微观层次之间并没有很明显的联系。这就导出了*隐秩序*的概念——系统中的不可识别的结构（Holland，1998）。这看上去像混沌，但实际上就是非线性行为。隐秩序可能就是范围的问题——那些近距离看起来不能识别的东西，但是当我们退后从远距离来看的时候就可能变得很明显了。例如，近距离观看一幅画的时候可能就是和那些随机的油漆污点差不多，但是我们从远距离看的话可能就是著名的蒙娜丽莎。那么涌现是按照比例因子变化的吗？

我们现在从更严格的角度去检验网络中的涌现。如图 7-1 中所示，动态网络的进化不会无缘无故地发生。主要有两个强制作用使它发生：（1）一系列内部微规则的反复应用；以及（2）外部环境力量的原因。前者是决定于*遗传或者内在的进化*，后者则是取决于*环境进化*。在这两种模式里，我们要观察从微观变化中涌现而来的宏观模式。在某些情况下，我们甚至可以通过长时间的微观进化来解释宏观行为。

7.2.3 生物中的涌现

在网络和动植物物种中的涌现是很明显的。实际上，一些当代的生物学家和自然历史学家

⊖ 我们实际上观察到随着随机网络中的密度接近 0% 或 100%，其熵会急剧下降。

声称生命本身就是涌现的产物——曾经被称为自发产生。通过长时间的非常小的称为变异的反复作用，生命就自发地诞生了。以无生命的化学物质和纯化学过程开始，通过漫长的进化，简单的生命有机体就生成了。

生物涌现需要我们相信复杂性的增长是以熵的减少为代价的。在每个微观步骤（化学反应）的应用之后，结构化替代了随机性。结构的演变是通过进一步的应用微规则使得更复杂的结构代替简单的结构。从某一点，无生命结构变成了有生命——复杂度到达了生命有机体的水平。这一过程持续、多样化，并且到达了更高层次的复杂结构。最终，达尔文的进化理论占据主导地位，导致了智慧的涌现。

从简单到复杂结构的涌现是在可控制的条件下演示的，但是还没有从非生命到生命的涌现的演示。有机物质可以从无机化学物质自发生成，但是有机化学物质到生命有机体的自动生成则有很长的一段路要走。作为科学家，我们必须对这一理论持怀疑的保留态度。

7.3 遗传进化

开环涌现源于网络自身内部。网络吸收能量并形成新的节点和链路，或者是对现有节点和链路进行重新安排。涌现是动态的——微规则每个时间步应用一次，最终导致了网络的显著改变。经过很长时间之后，如果涌现是收敛的话，网络将会到达一个终态。如果它是发散的，网络永远不会到达终态，而在有限或者无限的状态中循环。例如，假如有一个网络，有 n 个节点和 $m < n$ 条链路，它在每个时间步增加一条链路，直到网络变成完全的为止。当网络最终到达具有 $m = (n(n-1))/2$ 条链路的终态时，这个收敛过程将会结束。从另一方面来讲，在每个时间步增加一个新节点和新链路的网络是不会达到终态的。那么它就是发散的，会无休止地一直增加节点和链路。

遗传涌现其实很简单——反复地在每个时间步应用微规则并且观察结果。网络会收敛吗？在大多数情况下，我们猜想过足够长时间之后就应该会涌现一个特定模式。因此，我们可以检测一下“因果”假设。例如，如果我们不断地用较高度的节点取代较低度的节点，我们推测一个随机网络会演变成一个无标度网络。但是推断也可能是不正确的。实际上，下面的开环涌现的第一张图表，说明了这个推断是错误的！问题在于我们可以通过研究自然和人工合成系统的工作原理的因果解释来检测推断是否正确。

7.3.1 hub 涌现

考虑下面的开环涌现过程。最初，$G(0)$ 是一个具有 n 个节点和 m 条链路的随机网络。$G(0)$ 可以通过 ER 生成过程或者由前面描述过的锚定随机网络生成过程所创建。在每个时间步，随机地挑选一个节点和链路，同时问一个问题：“我们是否可以将这个随机选定的链路重联，使它可以连接到更高度的节点上？”在这种情况下，如果随机选定的节点的度比随机选定的链路所连的节点的度要高的话，就选定该节点作为重联的节点。链路被重联指向更高度的节点，或保留原样。

这个简单的微规则会一直重复下去。我们推断一个无标度网络会从这个随机网络中涌现出来，因为过了很长一段时间就会涌现一个高度的 hub。经过足够长的时间之后，$G(0)$ 的度序列分布会从泊松分布转换成幂律分布吗？我们将会在下面的分析中验证这个推测[⊖]。

“hub 涌现”微规则非常简单——只要能增加高度节点的度就重联随机选定的链路。只要用户需要，Network. jar 重复以下 Java 方法来实现 hub 涌现微规则：

⊖ 结果证明，推测是错误的。

```
public void NW_doIncreaseDegree(){
  //Rewire network to increase node degrees
  int random_node = (int)(nNodes * Math.random()); //A random node
  int random_link = (int)(nLinks * Math.random()); //A random link
  int to_node = Link[ random_link] .head;          //Link's to_node
  int from_node = Link[ random_link] .tail;        //Anchor node
  if(node[ random_node] .degree > node[ to_node] .degree){
     if(NW_doAddLink(node[ from_node] .name, node[ random_node] .name))
        NW_doCutLink(random_link);                 //Replace random link
     else  Message = "Warning: Duplicate Links Not Allowed.";
    }
  }//NW_doIncreaseDegree
```

这个过程是报酬增加律的另一个应用吗？在数千时间步之后，无标度网络会涌现吗？我们仅查看经过160 000个时间步涌现的度序列来检验我们的推测！图7-2a显示了随机网络 $G(0)$ 的度序列分布，图7-2b显示了经过160 000次迭代之后的分布。如果收敛网络 $G(160\ 000)$ 进化成无标度网络，它的度序列分布应该满足幂律分布。显然，并非如此。图7-2b显示了一个偏态泊松分布。最小的度节点有一条链路，最大度节点有178条链路，分布的峰值是在4条链路处。与随机网络相比：最大hub带有18条链路，最小hub带有0条链路，峰值为10条链路。

图7-2a的随机网络的度序列分布非常接近于平均值为6的泊松分布，但是它的 x 轴向左移了4个单位：Poisson(6,degree − 4)，图7-2b所示的hub涌现收敛网络与平均值为4的泊松分布非常接近：Poisson(4,degree)。不是将随机网络转换成无标度网络，hub涌现在随机网络中重新布置链路，这样一来度序列分布技术上仍是随机的，只是有一点偏移。在涌现之后，平均会有更多的小度节点，但是收敛网络会有度更高的hub。

具有最大链路数的hub——所谓的最大hub——随着时间 t 的平方根增长：$O(\sqrt{t})$，如图7-3所示。具体来讲，对于 $n = 100$，三个明显不同的密度为5%、10%、20%，hub涌现网络中的最大hub按照如下近似增加其度属性：

$$\text{hub\_degree}(G) = A + \frac{\sqrt{t}}{2}$$

这里，当密度为5%、10%、20%时，A 分别等于7、14和24。

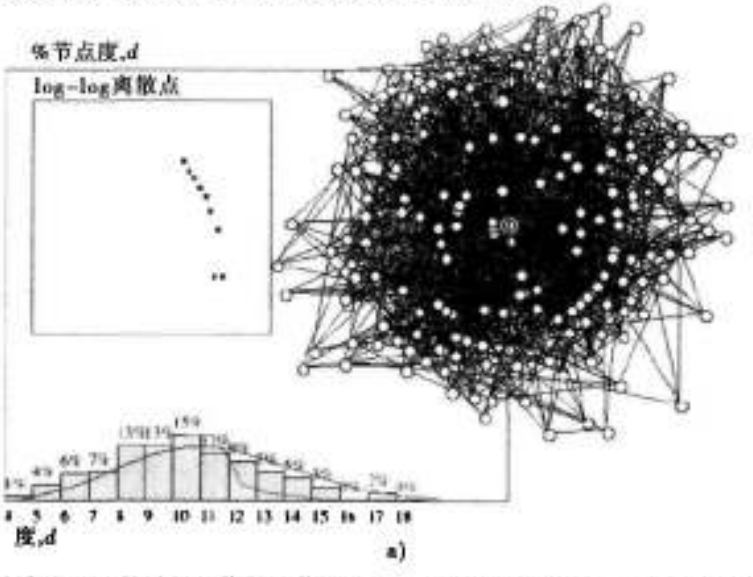


图7-2 具有高度hub的随机网络的网络涌现：a）随机网络 $G(0)$ ——涌现之前的度序列分布；（b）hub涌现网络 $G(160\ 000)$ ——涌现之后的度序列分布。初始网络 $G(0)$，$n = 200$，$m = 1000$，通过ER生成过程产生

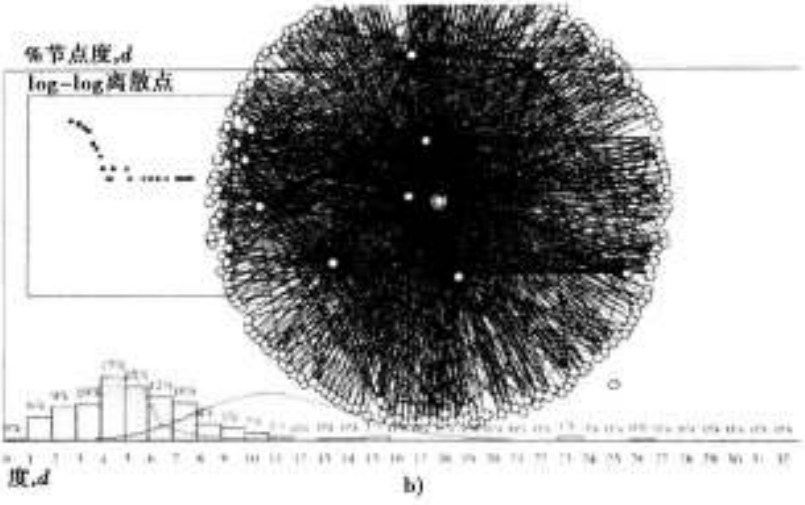


图 7-2 （续）

随着网络进化，最大和最小 hub 度之间的增长幅度是最明显的。最初受到随机网络密度的限制，扩展是随着时间的平方根而增长的。这种扩展对涌现网络的宏观属性有影响：熵、平均路径长度和聚类系数。

这又产生了另外一个问题："随着时间的推移，涌现网络的熵发生了怎样的变化？"回忆一下，熵是通过将从最小度到最大度的序列分布上的 $h(d)\log_2(h(d))$ 相加求和算出来的。扩展越大，求和的项目就越多。直觉上讲，它应该随着能量进入增加节点的度而增加，但是会随着 $h(d)$ 的减少而减少。但是实际情况是哪一个？

图 7-4 说明了随着前几条链路被重联到高度节点时，熵会迅速增长，但是随着随机性的减少会逐渐降低。换句话来讲，涌现网络首先由增加的随机性所主导，然后在达到一个拐点 t^* 后，就由增加的结构性（非随机性）所主导。hub 涌现规则的反复应用向网络注入了随机性，因为微规则会使重联随机化。hub 结构涌现并快速地控制着涌现网络的宏观结构，因为重联是偏好连接，而不是随机的。因此，涌现网络是随机和非随机力量结合起来的重构。

我们从第 4 章中知道熵以指数增长到一定点之后，然后当由结构接管时以指数形式减少到 0。另外，我们通过对无标度网络的研究知道，随着网络 hub 度的增加，随机性就会降低。涌现网络的结构是这两种竞争力量平衡的结果。

凭直觉我们有了关于熵的近似，如图 7-4 虚线所示。(1 - exp( )) 描述了之前增长的随机性，hub_degree( ) 描述增长的结构，它决定了一个越来越大的 hub 涌现。其中熵是不相同的

$$10(I(G)) = A + B(1 - \exp(-Ct)) - D\,\text{hub\_degree}(t)$$

在密度为 10% 时，这里的 $A = 37$，$B = 11.75$，$C = 0.00284$，$D = 0.141$。

用 sqrt(t)/2 替代 hub_degree(t)之后，对涌现网络的熵随时间的变化产生了一个近似：

$$I(G) = \frac{A + B(1 - \exp(-Ct)) - D \times \text{sqrt}(t)}{10}$$

当 $I(G)$ 达到最大值时，t^* 就是拐点。$I(G)$ 随 t 不同就导致一个关于 t^* 的非线性方程。这个问题就留给读者作为练习。

涌现网络经历两个阶段：在 t^* 之前和之后，这里 t^* 是熵达到最大值时的时间点。时间足够的话，涌现网络的熵会回到它的初始值，并且趋于 0，因为结构化最后完全主宰了随机性。理论上来讲，随机网络转化成结构化网络。但是在实际中，微规则会阻止这种事情发生，因为微规则只是朝向一个方向进行重联——每条链路的头部。

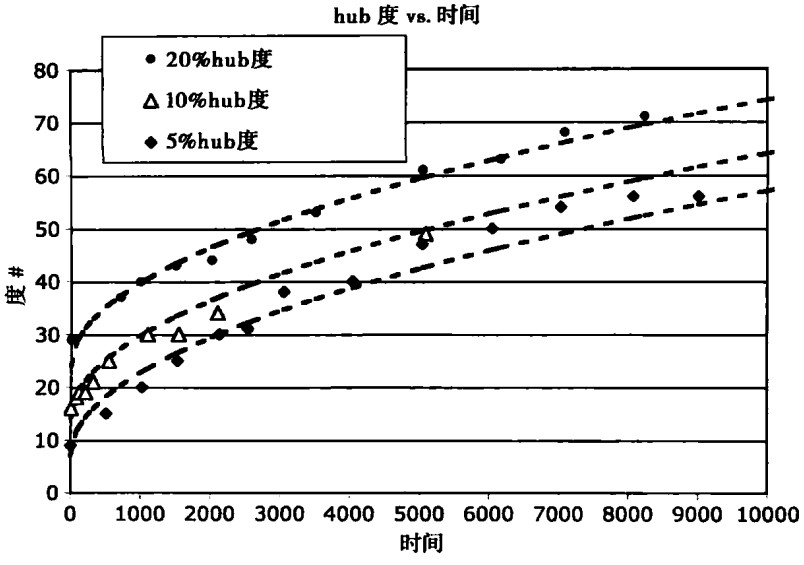


图 7-3 当图 7-2 所示的网络涌现时，最大 hub 度随时间而增长：$n = 100$，$m = 249$（密度 = 5%），$m = 500$（密度 = 10%），$m = 1000$（密度 = 20%）

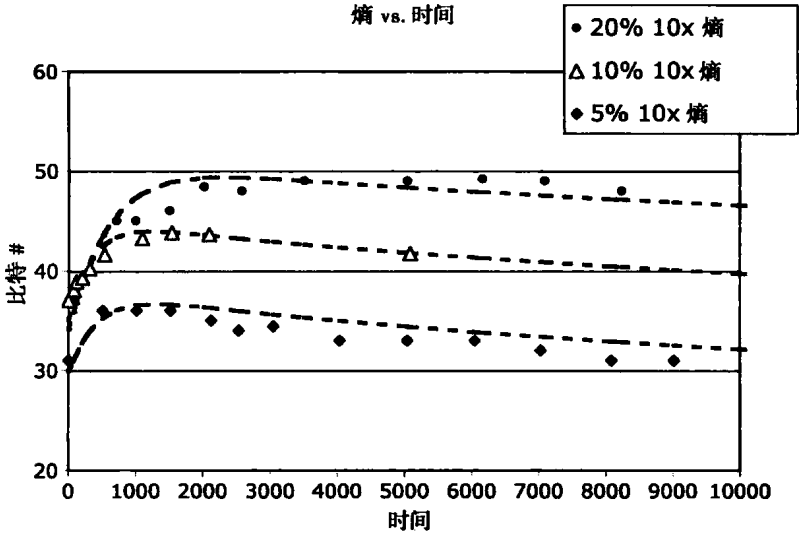


图 7-4 熵随时间变化。结构化网络 G（160 000）是从随机网络 $G(0)$ 涌现的，通过两个阶段：（1）进一步随机化；随后（2）逐渐主导了 hub 结构

链路的尾部从来都不会重联。度序列分布从一个窄的对称泊松分布转换为宽的偏态泊松分布，而不是瓦解变成一个尖柱状。

随着时间的推移，更大的 hub 逐渐形成，像预期的一样涌现网络的平均路径长度减少了。读者可以验证一下路径长度的减少是 $O(\text{hub\_degree})$，与时间成非线性的关系，因为 hub_degree 是 $O(\sqrt{t})$。相似地，聚类系数和时间的关系就留给读者作为练习。它是 hub 大小的函数，以 $O(\sqrt{t})$ 进行增长。

开始被认为是将随机网络转换为无标度网络的一种新颖的方法，现在结果证明是产生了一个全新类别的网络：带有 hub 的随机网络。这并不会产生矛盾，而是综合了二者属性的一种混合类。hub 涌现将随机网络转换成了一个带有 hub 的随机网络。对于这个新型的网络来说，对导航时间的影响是什么？这样的混合存在于自然界中吗？这些还未解决的问题，是写这本书时很热门

的研究课题。

7.3.2 聚类涌现

hub 涌现导致了带有 hub 的非无标度网络结构。那么有没有可能构造一个带有高聚类系数的非小世界网络？答案是肯定的，我们接下来会加以说明。再次以随机网络开始，假设我们使用如图 7-1b 所示反馈循环涌现以增加涌现网络的聚类系数。经过每个时间步之后我们保证网络的所有聚类系数不少于前一个时间步的。随着时间的推移，网络的聚类会增加——至少在理论上如此。

聚类系数涌现的原理如下。随机地选择链路和节点，如果总体的聚类系数保持不变或增加了，将链路重联到一个新的（随机）节点上去，如果因为重联而导致聚类系数下降的话，返回到前一个时间步的拓扑结构。无限地重复这个微规则，直至停止为止。

下面所示的 Java 方法 NW_doIncreaseCluster 通过一些预先判定措施精确地实现该工作。它随机地选取一条链路和一个节点，保证链路的首、尾和随机节点是不相同的。然后它调用 NM_doCC( )计算网络的聚类系数。如果聚类系数减少了，就返回减少的值；否则将网络的结构恢复到前一时间步的配置。这个方法同时也避免了重复的链路，在放弃之前尝试了 n 次连接来找到不重复的链路。

```
public double NW_doIncreaseCluster(double cc){
  LayoutReply reply = new LayoutReply();  //CC type
  double after_cc = 0;                //After link is changed
  int random_link = 0;                //Select a random link
  int random_node = 0;                //Select a random node
  int tail_node = 0, head_node = 0; //End nodes of random link
  int counter = 0;
  boolean done = false;
  while(!done && counter < nLinks){           //Avoid duplicate links
    random_link =(int)(nLinks*Math.random());//Select link at random
    tail_node = Link[random_link].tail;
    head_node = Link[random_link].head;      //End nodes of random link
    random_node =(int)(nNodes*Math.random()); //Select node at random
    while(tail_node == random_node || head_node == random_node)
      random_node =(int)(nNodes*Math.random()); //Make sure they differ
    //Trial
    done = NW_doAddLink(node[tail_node].name,node[random_node].name);
    if(done) {
         NW_doCutLink(random_link);    //Remove original link
         reply = NW_doCC();            //Find CC after adding new link
         after_cc = reply.mean;        //New CC
    }
    else  Message = "Rewiring Failed: Duplicate Link.";
    counter++;
  }
  if(counter >= nLinks){
    Message = "Rewiring Failed: Cannot find a node to connect.";
    after_cc = cc;
  }
  else if((after_cc < cc) && cc != 0) {
   NW_doCutLink(random_link); //Revert back
   NW_doAddLink(node[tail_node].name, node[head_node].name);//Restore
   after_cc = cc;                      //No change
   Message = "Revert to previous structure."+
     Double.toString(after_cc);
  }
  else Message = "Increasing Cluster Coefficient..."+
    Double.toString(after_cc);
  return after_cc;
}//NW_doIncreaseCluster
```

回顾第2章中的内容，聚类系数是所有节点 v_i 的聚类系数的平均值：

$$CC(G) = \sum_{i=1}^{n} \frac{C(v_i)}{n}$$

$$C(v_i) = \frac{2c_i}{\text{degree}(v_i)(\text{degree}(v_i) - 1)} \sim O\left(\frac{1}{\text{degree}(v_i)^2}\right)$$

这里的 c_i 等于连接到节点 v_i 的邻居节点的链路数。

有关该公式一定要记住的是，$C(v_i)$ 与节点度的平方成反比。随着度的线性增长，节点的聚类系数以幂律分布减少。该事实常用来解释当我们多次地将 NW_doIncreaseCluster() 方法作用在一个初始随机网络时所观察到的涌现。当涌现网络的 hub 度增加时，hub 聚类系数减少。减少是倒数关系 $O(\text{degree}(v_i)^2)^{-1}$ 的直接结果——不必是因为邻接节点之间更少的链路所造成的。

随着时间的推移，随机网络 $G(0)$ 会发生怎样的变化？会涌现小世界网络吗？如果你对小世界网络的定义等同于高聚类系数，那么涌现的网络就是小世界。如果聚类系数的定义与 Watts-Strogatz 定义是等价的，那么答案就是“不会”。

图7-5描绘了三个聚类系数涌现网络在5%、10%和20%密度下产生的聚类系数随时间的变化情况。所有的聚类系数都是在10% ~65%范围。聚类系数的增长是随时间递减的指数函数：$CC(G) = A + B(1 - \exp(-Ct))$，这里 A、B 和 C 都是常数，图7-5中描绘出的实验数据在表7.1中给出。另外，我们将等价密集的小世界网络的聚类系数列出来以便进行比较。数据和递减指数近似能很好地拟合，但是和同等密度的小世界的聚类系数差别很大。但是，平均路径长度却是在小世界和聚类系数涌现网络之间的。

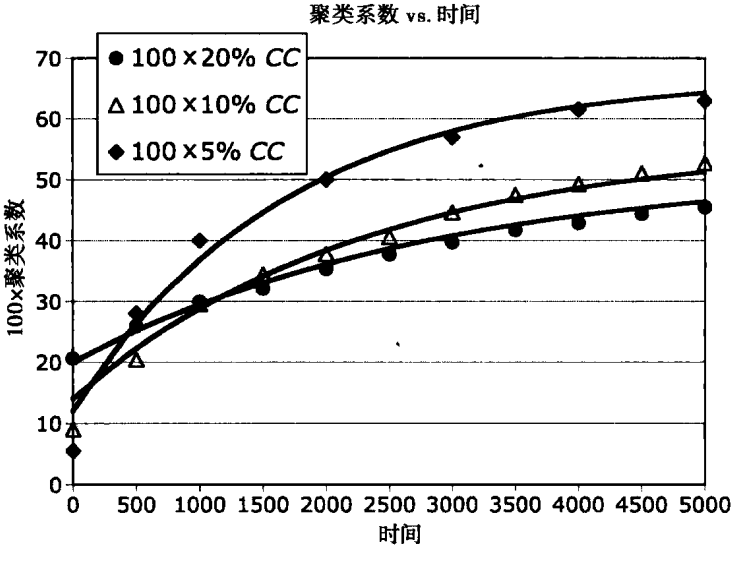


图7-5 $n = 100$，$m = 249$（5%），$m = 500$（10%），$m = 1000$（20%）时聚类系数随时间的变化

表7-1 对聚类系数参数 A、B、C 的曲线拟合近似

	5%	10%	20%
A	0.12	0.14	0.20
B	0.55	0.42	0.32
C	0.00060	0.00043	0.00035
小世界 *CC*	0.037	0.095	0.200

随着聚类系数的增长，涌现网络的熵、路径长度和 hub 度也会稍稍增长。这可能和直觉相矛盾。hub 度增长是最显著的，引出了对聚类系数增长的解释。我们发现，随着涌现网络的 hub 度的增长，其他非 hub 的度降低。根据方程 $CC(G)$，节点的度会随着聚类系数的增加而降低，因为 $CC(G) \sim O(1/\text{degree}^2)$。因此，许多低度节点的聚类系数增加，而少数高度节点的聚类系数降低。会有更多的低度节点具有高聚类系数，所以总体的聚类系数增加。

我们也许会有相反的意见，那就是每个节点周围的聚类的增加会导致全部平均值的增加。控制聚类系数涌现的微规则会重联链路以便增加局部聚类，这样带来的另外一个影响就是节点的度一般取值很广。随着聚类增加，度序列分布变平并加宽也支持了这样的解释（参见图 7-6）。

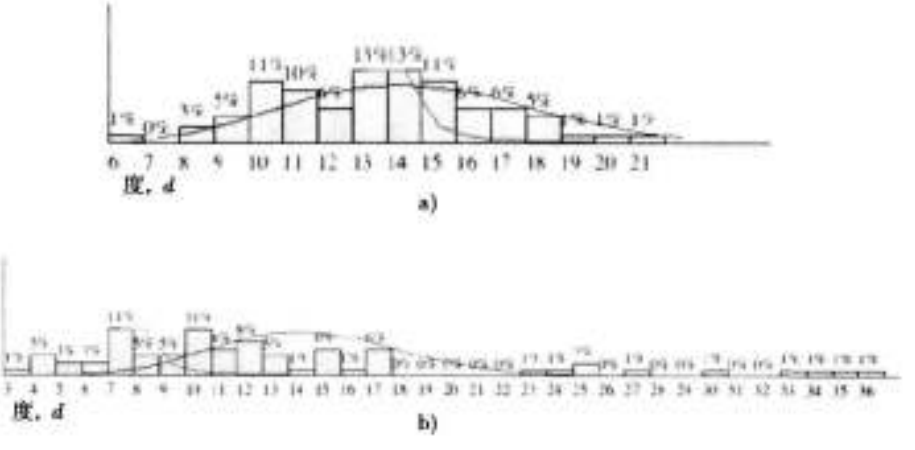


图 7-6 度序列分布比较：a）随机网络（$n = 60$，$m = 400$）；b）经过 $t = 10\,000$ 时间步之后聚类系数涌现的网络

7.4 设计者网络

我们已经说明了涌现是如何导致网络带有高度 hub 或者高聚类系数的。这些网络既不是无标度网络，也不是小世界网络。更确切地说，它们的度序列分布是伪随机的——具有某种如高度 hub 或大量聚类的全局属性。这就提出了一个问题："从任意一个初始网络中涌现一个具有特定高度序列分布的网路，可能吗？"换句话来说，我们可以通过指定各个节点的度来从随机网络中产生一个"设计者的网络"吗？

答案是"大部分时间"是可以的。给定度序列 g，我们就可以构造一个包含具有序列 g 的节点的网络，前提是这个拓扑是可实现的[⊖]。但是如果在节点对之间不允许重复链路的话，那么需要的网络就是不可实现的。在某些情况下，我们可能为了实用性（不重复）而牺牲精度来尽量接近。

在本节中，我们对节点度的总数进行限制，$\sum g = 2m$，这里 m 是 G 中的链路数量，我们使用偏好连接算法从任意起点产生任何可实现的网络。根据满足下面描述的限制的度序列 g 进行重联，度序列涌现会产生一个具有我们想要的精确拓扑的自定义网络或者称"设计者网络"。

我们使用方法 NW_doSetStates(total_value,mean_value)来存储想要的度序列 g，这里 g 是存储在初始网络 $G(0)$ 中的节点。当以初始状态或初始值存储在每个节点时，g 被称为剩余度或者剩余值。我们的目标是将 $G(0)$ 转变成 $G(t)$，以便 $G(t)$ 的度序列刚好就是 g。g 的初始元素在所需网络的进化中会减少，因此涌现网络刚好匹配 g。如果在进化之后所有的值都变成了 0，网络就收敛到了所需要的度序列 g。

⊖ 一个网络如果其度序列精确匹配 g，则称具有度序列 g 的网络是可实现的。

NW_doSetStates(total_value,mean_value)的第一个参数一般被设置成$2m$，因为每条链路连接两个stub，第二个参数设置成需要的网络平均度值，$\lambda = n/m$。如果total_ value小于$2m$，方法会报告一个错误消息，然后返回。如果mean_ value太大，网络将不会收敛，因为不能插入足够的链路以实现度序列g。

每个节点的状态s是介于一次最小值和两次最大值的平均度之间：$1 \leqslant s \leqslant 2\lambda$。最小度值保证了进化网络是连通的。任何节点的最大度都是$n-1$（$n-2$加上分配给所有节点的最小值1），因为不允许重复链路，所以每个节点最多可以连接$n-1$个其他的节点。

如果度的总数是奇数，那么至少还有一个度没有被使用，因为链路每次都是成对地减少度的。这个方法假设分配给所有节点的度的总数是偶数。因此，参数total_ value必须是一个偶数。下面的参数可以得到一个满意的结果：

$$\text{total\_value} \geqslant 2m, \text{一般是 } 2m$$

$$\text{mean\_value} = \frac{n}{m} \geqslant 1$$

为涌现做准备，方法NW_doSetStates()以g加载初始网络$G(0)$。除了对total_value和mean_value做了限制外，方法NW_doSetStates()必须保证有偶数个stub，stub的数量要小于最大可能值$n-1$，同时还要处理极端边界值n和m。

```
public boolean NW_doSetStates(int total_value, int mean_value){
  if(total_value < 2*nNodes || nNodes < 2
       ||
     mean_value < 1 ){
     Message = "Not Enough nodes or total_value < 2n. Change Preferences";
     return false;
  }
  if(!(total_value == 2 * (total_value/2))) total_value = 2*(total_value/2);
  Message = "States Set: Total = "+Long.toString(total_value);
  int sum = 0;
  for(int i = 0; i < nNodes; i++){
    node[i].value = 1;              //Minimum residual degree
    total_value-;
    sum++;
  }
  int count = 0;
  while(total_value > 0 && count < 10*nNodes) {
  for(int i = 0; i < nNodes; i++){
    if(total_value > 0){
       int s = (int)((2*mean_value) * Math.random());
       if(s > total_value) s = total_value;
       if(s >= nNodes - 2) s = nNodes - 2;      //Cannot be more than complete
       if(node[ i].value+s < nNodes-1) {
          node[i].value += s;
          sum += s;
          total_value -= s;
       }
        count++;
    }
  } }
  if(!(sum == 2*(sum/2))){
    for(int i = 0; i < nNodes; i++){
      if(node[ i].value > 1){
         node[i].value- -;
         break;
      }
    }
   }
   return true;
}//NW_doAddStates
```

使用该方法来设置 $G(0)$ 的初始度序列值 $g(0)$，这又称为设计者网络的目标值。在某些涌现过程中，每个节点的目标值或状态会减少直到0为止。那么，状态又被称为每个时间步的剩余度序列 $g(t)$。如果所有的节点达到0，那么设计者网络就实现了。否则，目标度序列是不可实现的。

7.4.1 度序列涌现

度序列涌现尝试将一个任意网络转变成具有规定度序列的网络。给定任意一个起始网络 $G(0)$，度序列 $g=\{d_1,d_2,\cdots,d_n\}$，这里的 d_i 是节点 i 的度，网络 $G(\text{final})$ 进化到具有度序列为 g 的网络。对于 $G(0)$ 中的每一个节点 v_j，目标度序列 g 以 value $(v_j)=d_j$ 存储在网络中。当 $\sum g=2m$ 并且允许重复链路时，我们就讲 $G(t)$ 的度序列收敛于 g。当 $\sum g<2m$ 时，因为链路太多了，度序列就不收敛于 g。如果 $\sum g>2m$，涌现要么停止，要么就能找到很多对 g 的近似。当 $G(\text{final})$ 是可实现的时，度序列涌现通过输入 g 来产生一个指定的“设计者”网络。

度序列涌现的目的是减少在每个节点存储的状态或值的 $G(0)$ 的初始度序列和指定度序列 g 之间的区别。这是有可能的：通过重联那些连接到带有太多链路的节点上的链路，因而减少每个节点的度直到它达到或者是近似于在 g 中指定的值为止。我们通过断开链路的一端或者两端，并将断开的链路重新连接到一个随机选定的节点上来实现。我们通过故意将链路连接到带有链路不足的节点上，特地不去尝试增加节点的度。

例如，假设 $G(0)=\{N(0),L(0),f\}$，$n=4$，$g=\{2,2,1,3\}$。注意 $\sum g=2+2+1+3=(2)(4)=8=2m$。图7-7a表示涌现之前的 $G(0)$，图7-7b表示经过100个时间步之后的 $G(100)$。刚开始，度序列 $g=\{2,3,2,1\}$，但是经过大约100个时间步之后，网络拓扑结构变成了所需要的：$g=\{2,2,1,3\}$。涌现缩小了 G 的初始度和目标度之间的差距。

我们建议一种简单的微规则：在有限时间里可以达到或者近似于 g。实际上，共研究三种方法：一种是涌现技术，一种是 Molloy 和 Reed 首先提出的早期方法（Molloy，1995），一个是近期被 Mihail 等人改进了的方法（Mihail，2002），只有最新的 Mihail 等人提出的方法在对 g 设置了限制的条件下可以确保收敛。

在下面，*L. head* 表示由链路 L 的首部所指向的节点，*L. tail* 表示链路 L 的锚定节点。链路和节点是随机选定的，重联必须要缩小实际的度和度序列值 g 之间的差距。下面给出的度序列涌现的微规则将 g 作为输入，产生一个等于或者是近似于 g 的涌现网络。

度序列涌现（每个节点）

1. 将 $g=\{d_1,d_2,\cdots,d_n\}$ 存储在 $G(0)$ 中的节点 $\{n_1,n_2,\cdots,n_3\}$ 的值域，比如 $v_i=\text{value}(n_i)=d_i$。
2. 无限循环：
   a. 从 G 中随机选取链路 L 和节点 r。*L. head* 指向链路 L 的首节点，*L. tail* 连接到 L 的尾节点。
   b. 如果 degree(*L. head*) > value(*L. head*)，将 *L. head* 指向 r。
   c. 否则，如果 degree(*L. tail*) > value(*L. tail*)，将 *L. tail* 指向 r。

这个算法不会减少存储在每个节点值域中的目标度数，但会将随机选定的节点的实际度与目标值进行比较。因为我们只是转换那些增加节点度（永不减少度）的链路，所以算法是“从低度作用到高度的”。只有在链路数量很少的节点的度增加时，才会附带减少链路很多的节点的度。这种“片面”的偏爱重联方式防止了跳跃或振荡。

度序列涌现的 Java 方法必须进行一些额外的检查，但是基本上和上面的算法是相同的。方法 NW_doPerNodeValue()保证了随机选取的节点是不同的。随机选定的节点 r 必须和经过链路 L 的节点是不同的，同时 r 必须具有一个大于0的值。方法 NW_doAddLink()禁止了重复链路，也就是说我们不能保证收敛。在一些情况下，这种方法只能得到一个近似的度序列为 g 的涌现网

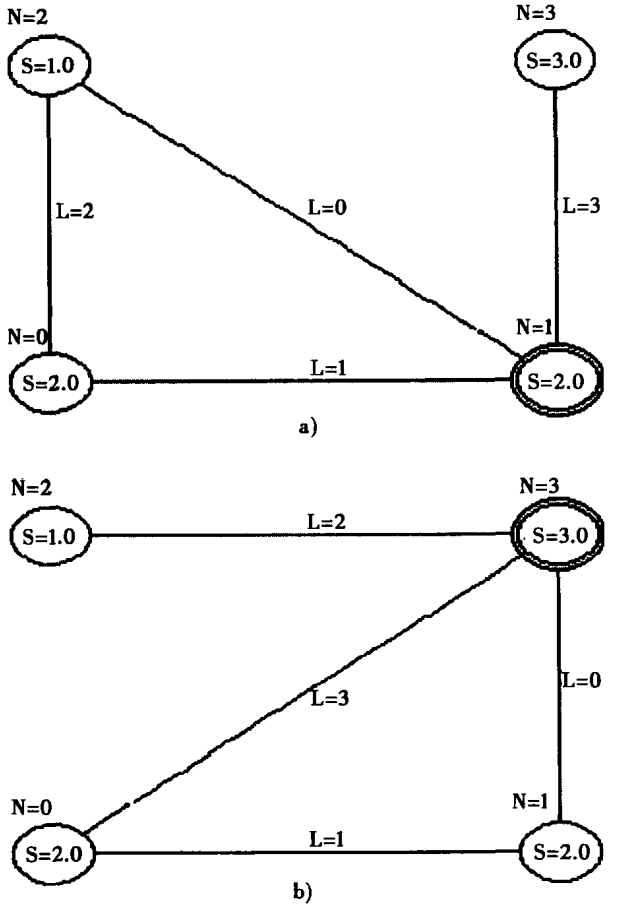


图7-7 设计者网络涌现之前与之后对比：a）之前，$G(0)$（$n=4$，$m=4$，$g\{2,3,2,1\}$）；b）之后，$G(100)$（$n=4$，$m=4$，$g\{2,2,1,3\}$）。d_j值以节点的状态值记录在节点内（$S=2.0$，$S=2.0$，$S=1.0$，$S=3.0$）

络。允许重复链路会有什么影响，这个问题就留给读者作为练习。

```
public void NW_doPerNodeValue() {
  int r = (int)(nNodes*Math.random());              //Pick a random node
  int random_link = (int)(nLinks*Math.random());    //Pick a random link
  int head_node   = Link[ random_link] .head;       //Pick an end node
  int tail_node = Link[ random_link] .tail;         //Rewire one or the other
  while(r == head_node
          ||
        r == tail_node
          ||
        node[r].value == 0)                          //Skip isolated nodes
      r = (int)(nNodes*Math.random());
  if(node[ head_node] .degree > node[ head_node] .value){ //Rewire head
    if(NW_doAddLink(node[ tail_node] .name, node[ r] .name, Link[ random_link] .name))
        {NW_doCutLink(random_link);}
    }
  else if(node[ tail_node] .degree > node[ tail_node] .value){ //Rewire tail
    if(NW_doAddLink(node[ r] .name, node[ head_node] .name, Link[ random_link] .name))
        {NW_doCutLink(random_link);}
    }
}//NW_doPerNodeValue
```

7.4.2 生成给定的度序列的网络

前面提出了有关特定拓扑网络的创建问题。我们现在知道如何产生一个随机网络、无标度网络及小世界网络，但是能产生给定度序列的网络吗？Molly 和 Reed 说明了如何通过使用算法（像上面描述的方法）来实现（Molloy，1995）。Molly-Reed 算法概括了正确的思想，但是它有一个致命的弱点——不能每次都保证想要的拓扑。

MR 算法在每个节点 v_i 创建指定值为 d_i 的 d_i stub（缺少另外一个端点的半链路，又称单臂链接）。然后，它执行一种偏好连接选择过程，将每个 stub 的松散末端连接到在另外一个节点上的松散末端。因为链路必须连接节点对，所以我们假设 $\sum g$ 是一个偶数。不断地重复连接 stub 的过程，直到所有的 stub 都被连接起来，或者直到所有可用的节点已经被用完为止[⊖]。

Molly-Reed 生成过程

1. 创建带有 n 个节点和零条链路的 $G(0)$，$N = \{n_1, n_2, \cdots, n_n\}$，每个节点具有值域 $v = \{v_1, v_2, \cdots, v_n\}$。

2. 在 G（0）的每个节点 v_i 的值域中存储 $g = \{d_1, d_2, \cdots, d_n\}: v_i = d_i; i = 1, \cdots, n$。

3. 重复执行直到没有 $v_i > 0$：

   a. 对于 $v_i > 0$，$v_j > 0$，随机选择一个节点对（i，j）。

   b. 插入链路 $n_i \sim n_j$。

   c. 对 v_i 和 v_j 递减，得到（剩余）残值 $v_i - 1$ 和 $v_j - 1$。

MR 在 Nerwork. jar 中以方法 NW_ doMolloyRead（）来实现。首先，方法创建 n 个节点，并且在每个节点的值域中存储了 g 个元素。所有节点值的总和 $\sum g$ 必须是一个偶数。然后通过随机地选择带有将要被链接起来的 stub 的节点尝试将 $\sum g$ 个 stub 连接起来。最初，存储在每个节点内的值等于由 g 指定的初始剩余度。每次算法将一对 stub 转换成一条链路，剩余度就会减少 1。如果所有的剩余度都减少为 0，需要的网络就会涌现。

然而，不能保证所有节点的剩余度都会减少到 0。实际上，可能会有一个或者多个 stub 没有和另一个 stub 匹配形成完整的链接。在经过数次不成功的尝试之后，这就需要放弃偏好连接循环。这种对方法的限制就会防止当涌现行为不收敛时出现无限循环[⊜]。

方法 NW_doMollyReed( )也保证随机选择的不同的非零值节点。（注意方法中的循环 while(i = =j) { } ）。这样就防止了自循环链路，并保证了对具有未链接 stub 的节点进行偏好连接。当 $\sum g$ 为偶数并且允许重复链路时，这个方法就可以保证收敛到想要的网络。为什么呢？

```
public void NW_doMolloyReed(){
    NW_doCreateNodes(nInputNodes);                  //Nodes, no links
    NW_doSetStates(2*nInputLinks, nInputLinks/nNodes);
    int n_stubs = 0;                                //Total number of stubs
    for(int i = 0; i < nNodes; i++) n_stubs += (int)node[ i] .value;
    String m = "Molloy-Reed Network, #Stubs = "+Long.toString(n_stubs);
    int count = 0;
    while(n_stubs > 0 && count < 100*nNodes) {  //Prevent infinite loop
     int i = (int)(nNodes * Math.random());     //Random node
     if(node[ i] .value > 0){
        int j = (int)(nNodes * Math.random()); //Distinct nodes
```

⊖ 当禁止重复链路时，在没有足够的剩余度或剩余值大于零的节点对时，Molloy-Reed 方法就插入很少的链路。

⊜ 我们将任意无数的尝试限制为 $100n$，但是这或许太大了。

```
        while(i == j) j = (int)(nNodes * Math.random());
        if(node[ j].value > 0){
           if(NW_doAddLink(node[ i].name,  node[ j].name)){
              n_stubs -= 2;
              node[i].value- -;                    //Decrement residual values
                    node[j].value- -;
                    count = 0;
                 }
               }
           }
           count++;
         }
         Message = m + " Unused = "+Long.toString(n_stubs);
    }//NW_doMolloyReed
```

Molloy-Reed 过程也许不能够实现 g。如果网络的度序列刚好满足 g，那么这个网络就是可实现的。但是正如我们前面所见，如果没有重复链路的话，度序列涌现可能不会收敛。例如，经过一段时间的涌现，假设仅有两个节点的剩余度是非零的。更进一步假设它们已经被链接了。在下一个时间步，这两个节点不能和任何其他的节点相链接，因为其余所有的节点的剩余度都是零。但是如果我们允许重复链路，这两个非零的链路可以被多次连接，至少可以将其中一个的剩余度减少为0。

什么条件才能保证收敛？直观来讲，收敛需要将所有的 stub（单臂链接）都连接起来，有多个 stub 的节点应该比那些“用完 stub”的少数节点先被连接起来。这就需要对 Molloy-Reed 生成过程做一些修改，以使拥有最多 stub 链接的节点比其他的节点先被连接起来。

我们猜想先连接高度值的节点会导致收敛。Erdos 和 Gallai 说明了按节点度递减顺序排列的度序列 $g = (d_1 \geqslant d_2 \geqslant \cdots \geqslant d_n)$ 是可以实现的，前提是存在匹配 g 的网络（Mihail，2002），并且满足：

$$\sum_{i=1}^{k} d_i \leqslant k(k-1) + \sum_{i=k+1}^{n} \min\{k, d_i\};\ 1 \leqslant k \leqslant (n-1)$$

Erdos-Gallai 定理是这个修改的关键，在某些约束条件下会导致收敛到一个可实现的网络的度序列涌现，图 7-7b 说明了 Erdos-Gallai 定理。将 g 按降序排列，$g = \{3,2,2,1\}$，$1 \leqslant k \leqslant 3$，计算 Erdos-Gallai 方程的左、右两边产生表 7-2 的结果。

表 7-2 证实了所有的 k 值都是不相等的。第 4 列必须小于或者等于第 7 列，也就是说 Molloy-Reed 过程收敛并且想要的度序列为 g 的网络是可实现的。在这种情况下，列 4 和列 7 是相等的。

假设表 7-2 中的 $k = 1$，与图 7-7b 中的节点 3 是相对应的（开始的时候度是0，在经过链路分配之后，度为3）。节点 3 必须获得三条链路，通过连接到 G 中的前三个节点。在这种简单情况下，前三个节点是按照 g 的减序排列：节点 0（$d_2 = 2$）、节点 1（$d_3 = 2$）和节点 2（$d_3 = 1$）。因此，节点 3 的 stub（单臂链接）是最先满足要求的，这将节点 3 的 stub 值减少为0，其余的每个节点的 stub 值减少 1。现在我们得到网络 $G(1)$，并且 $g(1) = \{0, 1, 1, 0\}$。对余下的非零节点继续执行该过程，节点 0 和节点 1 有一条 stub 链接，因此我们就将节点 0 和节点 1 连接起来。这样就得到 $G(2)$，并且 $g(2) = \{0,0,0,0\}$。$G(2)$ 收敛到了我们想要的网络。

Mihail 和他的同事提出的 Mihail 生成过程逐步实现了 Erdos-Gallai 理论。其实想法很简单，但是算法有点难以处理，因为在每个节点被连接之后就要按（剩余）残值对节点进行降序排列。首先，与 MR 过程中一样，需要创建节点，分配 stub 链接值。接着，对节点按降序进行排序，并且根据（剩余）残值排序解决 stub 链接。像前面一样，要保证没有自循环链接或者是重复链路。

表 7-2 对图 7-7b 中的 Erdos-Gallai 定理的证明

i	k	d_i	$\sum_{i=1}^{k} d_i$	$k(k-1)$	$\min\{k,d_i\}$	$\sum_{i=k+1}^{n} \min\{k,d_i\}$
1	1	3	3	0	1	(1+1+1) =3
2	2	2	5	2	2	2+ (2+1) =5
3	3	2	7	6	2	3 (2) + (1) =7
4		1				

Mihail 生成过程

1. 创建有 n 个节点和 0 条链路的 G (0)：$N = \{n_1, n_2, \cdots, n_n\}$。
2. 在 G (0) 中的每个节点的值域中存储 $g = \{d_1, d_2, \cdots, d_n\}$：$v_i = d_i$；$i = 1, \cdots, n$。
3. 重复执行直到没有 $v_i > 0$：
   a. 根据节点的剩余度将它们按照降序排序：$v = \{v_1, v_2, \cdots, v_n\}$，所以有 $v_1 \geqslant v_2 \geqslant \cdots 0$。
   b. 选取当前具有最大剩余度 v_1 的节点，将它称为锚节点 n_a。
   c. 将锚节点 n_a 和后面的 k 个节点 $n_2, n_3, \cdots, n_k$ 中的第二高度值节点相联。假设 $v_2 > 0$，$v_3 > 0$，$\cdots$，$v_k > 0$。注意这里 k 是非零剩余度值节点的数量，不包含锚节点。
4. 如果 ($k < v_1 + 1$)，这里 v_1 是锚节点的剩余度值，序列是不可实现的；否则将 $v_2, v_3, \cdots, v_k$ 对应的节点按照降序进行排序，然后继续。

Mihail 生成过程确保了每次最高剩余度节点被连接到第二高的剩余度节点时 Erdos-Gallai 条件仍旧保持有效，因为在每次重联之前都会将节点根据剩余度值进行降序排列。算法的下一步总是根据 Erdos-Gallai 条件，首先处理最高剩余度节点。

但是，如果目标度序列是不可实现的，那么 Erdos-Gallai 条件可能不成立。对于确保可实现性来说，这是一个必要但非充分条件。只有在拓扑是可能的时候，网络才有可能准确匹配度序列 g。这就是说拥有剩余度值的节点的数量至少必须和最大剩余度是一样大的。换句话来说，如果 k 是在经过 t 个时间步之后剩余度为非零的节点的数量，那么具有涌现网络中的最大剩余度节点 n_0 必须有一个不超过 ($k-1$) 的最大剩余度值：

$$(k-1) < \text{value}(n_0)$$

注意涌现网络可能分成多个部分，因为无法保证优先链接高度节点产生一个连通的网络。这里的实现不检查隔离，而且隔离也不会预防可实现网络的涌现。不考虑强连通条件的话，度序列条件是可以实现的。

算法是按顺序执行的，需要 $n-1$ 步。排序算法至少是 $n \log_2(n)$，所以全部时间复杂度很高：$O(n^2 \log_2(n))$。对大网络来讲，涌现会很慢。这里提供的实现方法甚至更慢，因为它的排序算法是 $O(n^2)$。

Mihail 和他的同事提出的算法的 Java 实现方法 NW_doMihail()，如果设置状态参数为真的话，就初始化 $G(0)$；否则的话它使用由用户或者函数 NW_doSetStates() 提供的值。在两种情况下，涌现仅从节点开始，所有的链接都会首先从 $G(0)$ 删除。

算法从类 Shuffler 中调用 SH_sort() 按照剩余度值将节点按降序进行排序。这种特殊的排序实现是 $O(n^2)$，所以这种算法是 $O(n^3)$，而不是 $O(n^2 \log_2(n))$。可以通过使用更加快速的 $O(n\log(n))$ 排序算法来降低时间复杂度。这就留给读者作为编程练习。

Mihail 生成过程在 Network.jar 中实现，使用下面的输入值将随机目标度序列 g 分配给 $G(0)$，当链路数量 m 远小于 $(n(n-1))/2$ 时，这些值最适合于随机网络的自动生成：

$$\text{total\_value} = 2m$$

$$\text{mean\_value} = \frac{n}{m}$$

像前面一样，Java 实现进行额外的检查以便确保网络参数（如 nNodes 和 nLinks）是合理的。当它不能够达到 g 或者所有的剩余度已经减少为 0 时，那么方法终止：

```
public void NW_doMihail(boolean set_states){
     Shuffler g;
     Message = "Mihail Network not realizable. Change preferences.";
     if(set_states) {                              //Initialize G(0)
        if(nInputNodes < 2) return;
        if(2*nInputLinks > nInputNodes*(nInputNodes-1)) return;
        NW_doCreateNodes(nInputNodes);             //Nodes, no links
        if(!NW_doSetStates(2*nInputLinks, nInputLinks/nInputNodes))
           return;
        g = new Shuffler(nInputNodes+1);
     }
     else g = new Shuffler(nNodes+1);              //Existing network
     nLinks = 0;                                   //No links, yet
     int n_stubs = 0;                              //Total number of stubs
     for(int i = 0; i < nNodes; i++){
        n_stubs += (int)node[ i] .value;
     }
     boolean done = false;                         //Until residuals = zero
     while(!done) {
       int k = 0;                                  //Erdos-Gallai condition
       for(int j = 0; j < nNodes; j++){            //Load g(0) into G(0)
          g.card[j] = j;
          g.key[j] = (int)node[ j] .value;
          if(g.key[ j] > 0) k++;
       }
       g.SH_Sort(nNodes, false);                   //Descending order
       int i = g.card[ 0] ;
       int d = g.key[ 0] ;
       if(d <= 0) break;                           //Converged
       if(k-1 < d)  {                              //Not realizable
         Message = "Mihail Network not realizable. Change
           preferences.";
         return;
       }
       //Connect top nodes
       for(int j = 1; j <= d; j++){
          int head = g.card[ j] ;
          if(node[ head] .value == 0) {
             done = true;
             break;
          }
          if(NW_doAddLink(node[ i] .name, node[ head] .name)){
             n_stubs -= 2;
             node[i].value- -;
             node[head].value- -;
          }
       }
     }
     Message = "Mihail Network: #Stubs Unused = "+Long
       .toString(n_stubs);
}//NW_doMihail
```

7.5 排列网络涌现

到目前为止描述的微规则对度序列分布会有负面影响，可能会改变涌现网络的类型。另一

方面，排列变换是不会改变度序列分布的，但是会产生一个具有不同映射函数的网络。对链路进行排列，而不改变度序列，将会引起给定类型但聚类、路径长度等特定的网络涌现。

设 $\pi_0(G)$ 为 G 中链路的排列，度序列分布是不变的：

$$\pi_0(g(G(t))) = g(G(t))$$

对于所有的 t 来讲，这里 $G(t)$ 是具有度序列 $g(t)$ 的时变网络。

我们说在 π_0 下度序列 $g(t)$ 是 $G(t)$ 的一个固定点，因为固定点在转换中不会改变。但是其他的属性，比如网络的聚类系数、路径长度，都可能会改变。什么样的微规则保证了固定点排列变换？

考虑图7-8中所示的微规则。随机地选择两条链路：一条链路连接节点 $x - y$，另外一条连接节点 $u \sim v$。四个端节点是不同的，因此我们可以如图7-8中所示那样排列链路。在排列之后，就变成了 $x - v$ 和 $u - y$。x、y、u 和 v 的度仍然没有变化，因此度序列分布也没有变化。不管微规则执行多少次，都不会对度序列分布造成影响。因此，从这个微规则反复应用涌现出来的网络仍然是同一类网络，因为它的度序列没有发生变化。

图7-8说明了一个可能的排列——水平横向链路排列，即链路交换头节点。另外一个排列——垂直纵向链路排列——交换尾节点：一条链路的头节点与另外一条链路的尾节点交换。读者可以考虑其他不改变度序列的链路排列，因此也称为等价变换。例如，如果两条链路的尾节点交换之后会发生什么？像这样的交换也称为“调换”（flip）。

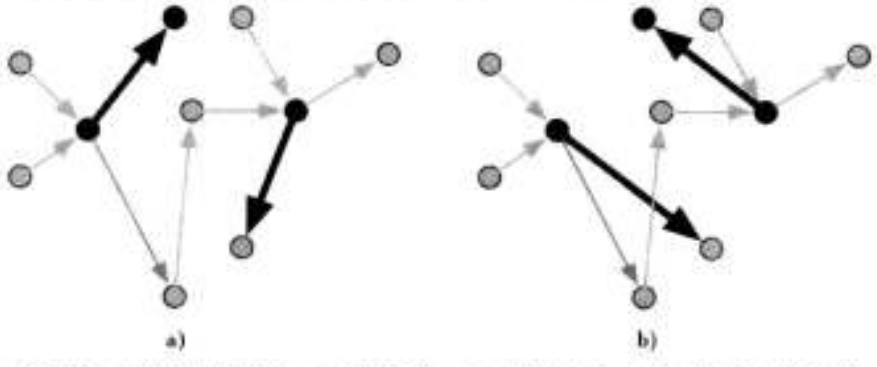


图7-8 排列或“调换”微规则：a）调换之前；b）调换之后。随机地选择两条链接（加粗显示），然后交换头节点。所有四个节点必须是不同的

7.5.1 排列微规则

链路排列并不是像图7-8中描绘的那么微不足道。算法必须要检查两个基本的冲突：（1）四个端节点必须是不同的，否则的话会改变度序列；（2）新的链路映射必须要避免重复链路。算法在调换链路时要避免重联到一个已经存在的链路上。这两个条件使得如下所示的Java代码要比预期的复杂得多。因此，我们将代码分成了两种方法——一种是对随机链路的选择，另外一种是对实际链接的调换操作。

方法 NW_doPermutation( )尝试找到2条链路，有4个不同的端点——直要找 nLinks 次才会放弃。仅只调换头节点，所以有向链路的含义保持不变（每个节点的入度和出度被保存）。在初始化网络 $G(0)$ 中至少要有4个节点和4条链路。

```
public void NW_doPermutation(){
  int link1 = (int)(nLinks * Math.random());
  int link2 = (int)(nLinks * Math.random());
  int count = 0;                //Try nLinks times before giving up
  while(count < nLinks
              &&
     (Link[link2].head == Link[link1].head
                 ||
      Link[link2].head == Link[link1].tail
                 ||
      Link[link2].tail == Link[link1].head
                 ||
      Link[link2].tail == Link[link1].tail)){
     link2 = (int)(nLinks * Math.random());   //Distinct end nodes
     count++;
    }
    NW_doFlipLinks(count, link1, link2);      //Flip, if no duplicates
    }//NW_doPermutation
```

方法 NW_doFlipLinks( )有一个更困难的任务——如果检查到有重复链路，它必须要准备放弃调换操作。如果两个节点已经连接了，那么方法 isConnected ( ) 就返回 true。如果 4 个节点是不同的，而且发现没有重复链路，那么就交换两条随机选择的链路头节点：

```
public void NW_doFlipLinks(int count, int link1, int link2) {
   if(count < nLinks) {                    //Flip links if possible
      int saved_head1 = Link[ link1].head; //Prepare to revert back
      int saved_head2 = Link[ link2].head;
      if(!isConnected(Link[ link1].tail, Link[ link2].head)
                        &&
        !isConnected(Link[link2].tail, Link[link1].head)) {
        Link[link1].head = Link[link2].head;        //Flip
        Link[link2].head = saved_head1;
      }
      else {                                  //Revert: no duplicates
        Link[link1].head = saved_head1;
        Link[link2].head = saved_head2;
      }
   }
   else Message="Warning: Cannot Permute Links.";     //Not possible
}//NW_doFlipLinks
```

整个过程中排列微规则不改变 $G(t)$ 的度序列，但是它可能会改变 G 的其他属性。例如，在不改变网络的度序列分布的情况下，增加（降低）某个网络的聚类系数、平均路径长度、中心性或者直径是可能的。排列变换的这个特性在多个应用中是很有用的，具体描述如下。

7.5.2 排列和聚类系数

排列变换 $\pi_D(G)$ 在创建带有目标度序列分布和其他属性（如聚类系数）的自定义网络方面是很有用的。例如，我们可以通过建造一个反馈循环涌现模型来最大化任意网络的聚类系数值。确切地说，在保持随机网络的随机性的同时增加它的聚类系数是可能的，并且在小世界网络已经有很高聚类系数的情况下增加聚类系数也是有可能的。正如我们将要看到的，一般来说在不改变度序列分布的情况下增加任意网络的聚类系数是可能的——但是有一定的上限[⊖]。

考虑下面的涌现算法，它在不改变网络类的情况下最大化聚类系数。下面的聚类系数排列算法将任意网络作为起始点 $G(0)$，并且产生一个具有最大聚类系数的网络 $G(\text{final})$。这个过程

⊖ 聚类系数受到网络链路数和网络密度的限制。

中会使用数千次的排列变换——注意仅保持增加聚类系数的排列，丢弃所有其他的排列：

聚类系数（排列）

1. 生成任何想要的度序列分布的网络 $G(0)$。
2. 计算初始网络的聚类系数：$c(0)=CC(G)$。
3. 无限循环：
   a. 应用链路排列微规则 $\pi_D(G)$。
   b. 计算网络的聚类系数 $c(t)=CC(G)$。
   c. 如果 $c(t)<c(t-1)$，撤销 $\pi_D(G):G(t)=G(t-1)$；否则什么也不做。

该算法和排列变换算法几乎是相同的的。实际上，加上聚类系数计算并丢弃不能够增加聚类系数的排列变换的反馈循环之后就完全一样了。如果连接了节点 v 和 w，那么方法 isConnected（v,w）就返回真。这一步会避免链路的重复，这是强加到我们网络拓扑之上的要求之一。像前面一样，排列可能会失败，所以变量 count 限制了尝试次数，以便防止无限循环。

正如我们预想的一样，Java 代码和 NW_doPermutation( )非常类似，它提供了这种实现的基础。NW_doPermuteCC( )将两种排列方法（水平横向和垂直纵向）合二为一，并且添加了不减少聚类系数的尝试：

```
public double NW_doPermuteCC(double cluster_coefficient){
  LayoutReply reply = new LayoutReply(); //Return value type
  reply.mean = cluster_coefficient;      //CC(t-1)
  int link1 = (int)(nLinks * Math.random()); //Permutation links
  int link2 = (int)(nLinks * Math.random());
  int count = 0;
  while(count < nLinks
        &&
    (Link[link2].head == Link[link1].head
                ||
    Link[link2].head == Link[link1].tail
                ||
    Link[link2].tail == Link[link1].head
                ||
    Link[link2].tail == Link[link1].tail)){
    link2 = (int)(nLinks * Math.random()); //Distinct end nodes
    count++;
}
if(count < nLinks) {                       //Flip links if possible
    int saved_head1 = Link[ link1].head;
    int saved_head2 = Link[ link2].head;
    if(!isConnected(Link[ link1].tail, Link[ link2].head)
                  &&
      !isConnected(Link[link2].tail, Link[link1].head)) {
        Link[link1].head =          Link[link2].head;     //Flip
        Link[link2].head = saved_head1;
        reply = NW_doCC();                                //CC(t)
        if(reply.mean >= cluster_coefficient) return reply.mean;
        else {                                 //Revert due to no increase
        Link[link1].head = saved_head1;
        Link[link2].head = saved_head2;
        return cluster_coefficient;
      }
  }
  else {                                  //Revert due to duplicate links
    Link[link1].head = saved_head1;
    Link[link2].head = saved_head2;
    return cluster_coefficient;
  }
```

```
}
else {
 Message = "Warning: Cannot Permute Links.";          //NotPossible
   return cluster_coefficient;
   }
}//NW_doPermuteCC
```

图 7-9 总结了使用通过 NW_doPermuteCC( )来实现的排列微规则和聚类系数反馈循环的几种类型的网络进化结果。图 7-9 的数据是通过 Network. jar 计算出来的，Network. jar 针对稀疏网络，并且有 10 000 个循环。从图 7-9 中我们可以推导出下面结果：

1. 从最小到最大聚类系数排序——小世界网络在起点附近发生相变之后，三种网络类型的排序如下：随机网络、小世界网络和无标度网络。

2. 聚类系数 $CC(G)$ 在三个阶段中单调递增：初始化、速率效应和 hub 效应。聚类系数是作为这些阶段的总和来被建模的：

$$CC(G(t)) = A + B\exp(-Ct) + D\log(Ct)$$

这里 A 对应于初始化阶段；$B\exp(-Ct)$ 对应于速率效应阶段；$D\log(Ct)$ 对应于 hub 效应阶段。表 7-3 列出了从图 7-9 中获得的 A、B、C 和 D 的值。

对图 7-9 的首要发现并不让我们感到奇怪——随机网络要比同等规模的无标度网络和小世界网络的聚类小很多。令人惊讶的结果是在稀疏无标度网络中的聚类系数增长很快。从第二步开始，无标度网络的增加速度远远超过了小世界网络。为什么？

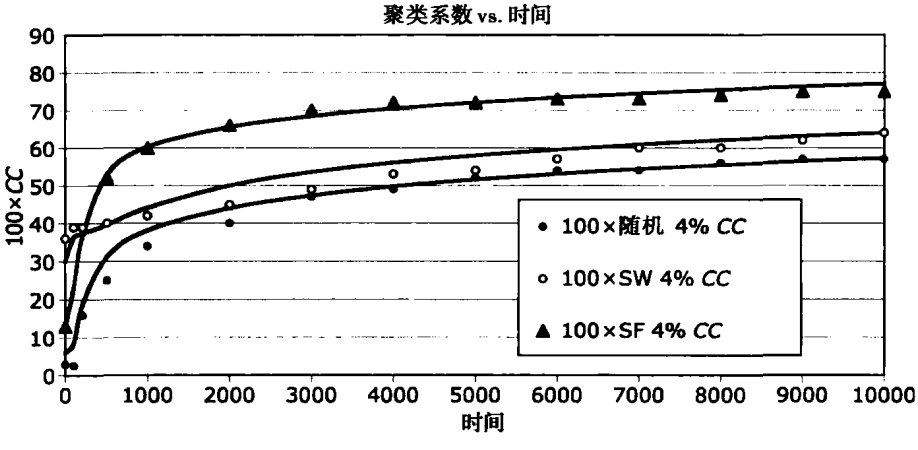


图 7-9 在密度近似为 4% （$n = 100, m = 200$），经过 10 000 循环之后，随机网络、小世界网络和无标度网络的最大聚类系数

三种网络中聚类的增加是由于三个因素，随着时间的推移，每个因素都在时间间隔内变成主导地位。第一个因素就是固有的聚类，这取决于网络类型。参数 A 代表了这个因素，并且它在简短的初始阶段占主导地位。表 7-3 说明了小世界网络在三个网络中的固有聚类系数是最大的。它的参数 A 值是无标度网络的 2.5 倍，是随机网络的 5 倍。随机网络和无标度网络的初始聚类都很小。

进化的第二阶段是由速率效应主导的——聚类系数的增加速率，受到已经建立的聚类量的抑制。网络的聚类越多，获得更多聚类的速率就会更低；也就是 $\delta CC/\delta t \sim \alpha(CC)$，这里的 $\delta CC/\delta t$ 是聚类系数的变化率，$\alpha < 0$ 是一个速率常数。对这个速率方程的解为 $CC \sim \exp(-Ct)$，这里 $\alpha = -C$，为表 7.3 中的第三个常数[⊖]。

在速率阶段，聚类涌现是由网络中的聚类量（要么缺少聚类）所主导的。随着聚类的增加，

⊖ C 为时间压缩或速率常数。

额外的聚类更难获得。因此，速率随着时间以指数方式减小并消失。参数 C 对三种网络来说是一样的，因为 n 和 m 是一样的。

第三阶段由 hub 效应所主导——聚类在小 hub 附近比大的 hub 附近更容易增加。我们在前面已经观察过该效应，聚类在低度节点出现的可能性要比在高度节点出现的可能性大，因为节点要增加 $O(d^2)$ 条邻接链路以与 $O(d)$ 成比例地增加它的聚类系数。因此，在有很多低度节点的网络中增加整体的聚类系数要比在有很少低度节点的网络中容易一些。这样一来，无标度网络中低度节点的数量是三种网络中最多的，所以它的 hub 效应最大。

在速率效应阶段，无标度网络快速获得聚类，因为它开始的时候聚类节点数是最少的。所以，它具有最大的参数值 B。在 hub 效应阶段它仍然会继续增加聚类，而不是以其他两个的同样速率，因为它具有更大的 hub。更大的 hub 会抑制聚类系数的增长，这是因为高度 hub 需要很多邻接链路。

随机网络和小世界网络类也会被相似的 hub 效应所影响。小世界的参数 D 要稍微大一点，这是因为小世界的度序列分布往往比随机网络的要更窄一些。

表 7-3 对图 7-9 的曲线近似参数

参数	随机	小世界	无标度
A	6	30	13
B	19	20	36
C	0.005	0.005	0.005
D	19	20	16.5

如此一来，小世界 hub 节点的度要稍微比同等规模的随机网络的 hub 度低一些。

我们也可以把相似的逻辑应用到网络中的聚类涌现及其密度，如图 7-10 所示。在这个实验中，聚类允许在 3000 时间步涌现，密度变化范围是 2% ~ 50%。聚类系数的增加同样也是这三个因素的函数，另外再增加一个——网络密度。

随着密度的增加，四个因素，对应四个阶段，决定了随机网络、小世界网络和无标度网络的聚类。第四个因素是密度：

$$CC(G(d)) = A - B\exp(-Cd) - D\log(Cd) + ECd$$

这里 A、B、D 和 E 表示初始状态、速率、hub 以及密度的效应。

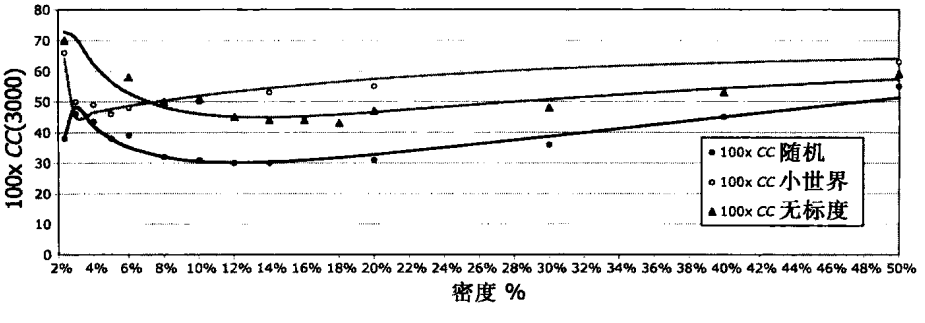


图 7-10 聚类系数涌现随着网络密度的变化。数据是在时间为 3000、密度为 100 × CC 的条件下绘制的

注意：

A = 初始的或固有的聚类系数

$B\exp(-Cd)$ = 速率效应

C = 时间常数

$D\log(Cd)$ = hub 效应

$E(Cd)$ = 密度效应

d = 密度

通过曲线拟合获得的所有参数值总结在表 7-4 中，这些值支持以下结论：

1. 正如我们对小世界网络的研究预期一样，随机网络和小世界网络在密度为 3% ~4% 的时候会经历一个相变。但是，在小世界网络中的聚类系数变化几乎和随机网络的变化是相反的，也就是说，其中一条曲线近似为另外一条曲线的镜像映射，按照水平线接近 CC（3000）=0.45。这是因为在小世界网络中将结构化转换成随机化，以及速率效应在随机网络中占主导地位。在两种情况下，密度都是决定聚类系数的最大因素。

2. 随着密度的增长，随机网络和小世界网络会经历四个阶段：初始聚类值（随机网络的很低，小世界网络的很高），速率效应（两种网络都很高，但是由于它们的初始聚类系数值，是以相反的方向），hub 效应（对小世界网络来说是正的，对随机网络来说是负的），密度（在随机网络中占主导地位，但在小世界网络中不存在）。

3. 对于无标度网络来讲，随着密度的增长，开始时聚类系数是减少的，然后开始增长，反映了速率和 hub 的均衡效应作用。在第二个阶段速率效应占主导地位，在第三和第四个阶段，hub 效应占主导地位。随着密度增长，无标度网络 hub 的度动态增长。因此，聚类系数主要是被 hub 的大小所制约，我们知道，对于无标度网络来说是按照对数增长的。

4. 在所有的网络中（随机、小世界、无标度），当密度接近 100% 的时候，聚类系数以相同的值开始涌现。这很明显，因为 100% 密度的网络是一个完全网络。完全网络的聚类系数是 100%。

图 7-10 揭示了一个惊人的结果——如果密度低于 8%，涌现的无标度网络的聚类系数比同等规模的小世界网络的聚类系数要高。小世界网络的聚类系数应该比无标度网络的高，我们甚至发现排列涌现可以将无标度网络进化为一个比同等小世界网络带有更多聚类的网络！我们只有通过查询表 7-4 才能够弄明白原因。无标度网络涌现的参数 B 和 D 要比小世界网络的高得多。这个令人印象深刻的事实完全是因为在无标度网络中低度节点的部分与小世界网络相比很高。换句话来讲，涌现产生的网络具有两个特性——高度 hub 和高聚类系数值。

表 7-4 对图 7-10 的参数拟合值

参数	随机	小世界	无标度
A	36	64	71
B	66	18	110
C	5	5	5
D	80	-4	112
E	21	0	16

7.6 涌现的一个应用

我们可以使用反馈涌现和适当选择的微规则来构造任何类型的网络，这项技术对很多路由和应用部署非常有用。作为一个具体的例子，我们会说明在二维空间里怎样最小化连接一组随机放置的组件所需的线路长度，比如说印刷电路板，或者是社区里相邻的房间。其思想很简单：建立一个基本的拓扑，比如说保证所有的组件都被连接，并且不断应用排列变换，直到达到我们想要的最小线路值为止。我们利用计算机来完成所有这些繁琐的求解方程的工作。

在第一个例子中，我们将锚节点或者 ER 随机网络分散到一个二维空间中，然后对链路对执行排列操作。我们只接受减少网络总长度的排列，其余的都不允许。当它实现的时候，链路的最小总长度 $\sum e_i$ 以指数方式收敛于一个正比于 $O\left(\frac{m}{\sqrt{n}}\right)$ 的值[⊖]。

接下来，我们对带有节点度限制的随机放置在表面的组件布局使用相同的排列方法。对于两条链路连接所有节点的情况下，degree $(v) = 2$，连接的最小长度仅是一个 1 - 规则或连接最近相连节点的环形网。对于更高度的网络，最小总长度的涌现将空间上最近的两个邻居连接起来。

7.6.1 随机排列的链接优化

考虑以下实验。创建一个有 n 个节点、m 条链路的网络，使用锚定随机网络或者 ER 随机网络生成程序[⊜]。

随机地将节点散布到表面。下面演示的这个方法使用随机极坐标来散布节点，注意停留在屏幕边界之内。任何将 n 个节点散布到 $\sqrt{n} \times \sqrt{n}$ 区域的方法都可以。

```
public void NW_doRandomizeNodes() {
   for (int i = 0; i < nNodes ; i++) {
      int r = 2*(int)(CircleRadius * Math.random());
      double arc = Math.random() * (2.0*Math.PI)/nNodes;
      node[i].x = x0 + (int)(r*Math.cos(i*arc));
      if(node[ i].x > xwide-40) node[ i].x = xwide-40;
      if(node[ i].x < 40) node[ i].x = 40;
      node[i].y = y0 + (int)(r*Math.sin(i*arc));
      if(node[ i].y > ytall-40) node[ i].y = ytall-40;
      if(node[ i].y < 40) node[ i].y = 40;
   }//for
}//NW_doRandomizeNodes
```

一般来讲，我们会反复应用排列变换。如果链路的总长度增加的话，就将排列变换丢弃，否则就保留。随着时间的推移，我们希望短的链路可以取代长的链路，如此最小化所有链路的总长度。最佳解是从不断地重复利用简单的排列涌现而来的。排列变换并不像看上去那样直截了当，它必须以很多方式受到限制：

1. 随机选择的链路必须是不同的——不允许出现三角子网。如果在尝试了 m 次之后无法实现，就放弃。

2. 要注意已经存在的链路——在调换过程中不能重复已经存在的链路。试着水平纵向调换链路，如果失败的话，就试着横着调换。如果都不行，那么就终止。

3. 要注意组件——如果网络的组件超过一个，就终止涌现进程。避免调换链路导致多个组件。

4. 如果步骤 1 ~ 3 都失败了，那么恢复到之前的拓扑，返回以前的总长度。否则，保留新的拓扑，返回短一些的总长度。

下面的 Java 方法将当前总长度作为参数，要么返回新的总长度，要么返回原始参数。因为它使用了链路排列，原来的网络拓扑被保存下来。因此，这个优化是固定点变换的一个应用。如果可能的话，通过将长的链路与短链路交换可以降低总长度。随着时间的推移，所有的短链路都被找到，所以总长度会达到最小值。

方法 isComponent (0) 将从节点 0 开始遍历整个网络，以判断网络是否是强连通。如果 $G(t)$ 中的每个节点从节点 0 出发都是可达的，那么就返回 true。如果排列使得网络断开的话，那么就丢

⊖ 这不是一个通用的结果，而是一个二维布局的人工制品。

⊜ 当 m 很小时，需要锚定随机网络，因为我们想要 $G(0)$ 是强连通的网络。大多数情况都是如此，但也有例外的。

弃这个排列。再次，如果调换链路对 m 次尝试都失败的话，那么微规则就会被终止。反复执行这个方法，直到收敛为止。在下一节中，我们估算一下需要多少次循环才能取得收敛以达到最优解。

```
public double NW_doShorter(double length) {
    String M = "+Shorten: Length = ";
    int link1 = (int)(nLinks * Math.random()); //Choose 2 links
    int link2 = (int)(nLinks * Math.random());
    int count = 0;                              //Avoid infinite loop
    while(count < nLinks
                          &&
       (Link[link2].head == Link[link1].head  //Distinct end nodes
                          ||
        Link[link2].head == Link[link1].tail
                          ||
        Link[link2].tail == Link[link1].head
                          ||
        Link[link2].tail == Link[link1].tail)){
       link2 = (int)(nLinks * Math.random());   //Distinct end nodes
      count++;
    }
    int saved_head1 = Link[link1].head;          //For revert...
    int saved_tail1 = Link[link1].tail;
    int saved_head2 = Link[link2].head;
    int saved_tail2 = Link[link2].tail;
    boolean flip = (count < nLinks);             //Feasible to flip
    if(!flip) return length;                     //Otherwise give up
    if(Math.random() < 0.5){                     //Duplicate?
        flip = (!isConnected(Link[link1].tail, Link[link2].head))
                                        &&
               (!isConnected(Link[link2].tail, Link[link1].head));
        if(flip){                                  //Flip vertical
            Link[link1].head = Link[link2].head;
            Link[link2].head = saved_head1;
        }
     }                                             //Duplicate?
     else { flip = (!isConnected(Link[link1].head, Link[link2].head))
                                          &&
                 (!isConnected(Link[link1].tail, Link[link2].tail));
            if(flip){                              //Flip horizontal
                Link[link1].head = Link[link2].head;
                Link[link1].tail = saved_head1;
                Link[link2].tail = saved_tail1;
                Link[link2].head = saved_tail2;
            }
     }
     if(flip){                                     //No duplicates
            flip = isComponent(0);                 //Strongly connected?
            double new_length = NW_total_length(); //Has it shortened?
            if(new_length <= length && flip){
               Message = M+Double.toString(new_length)+" Flipped Links.";
                return new_length;
            }
     }
     Link[link1].head = saved_head1;               //Revert
     Link[link1].tail = saved_tail1;
     Link[link2].head = saved_head2;
     Link[link2].tail = saved_tail2;
     Message = M+Double.toString(length)+" Warning: Cannot Permute Links.";
     return length;
}//NW_doShorter
```

7.6.2 确定性排列的优化

由于链路对的随机选择的不可预见性，前面提到的涌现过程似乎是在浪费努力。有没有确定性的算法可以更快地找到最小总长度？它能模仿自然过程吗？我们接下来会说明一个确定性的算法，它花费的时间几乎是相同的，但是不代表自然过程，因为它使用的是一个有目的的或者是确定性的启发式搜索。

通过简单地列举出所有可能的链路组合，最小总长度可以经过正比于 $C_2^m = m((m-1)/2)$ 的时间被找出来。在链路对上重复的趟数会减少长度目标函数并且保证是最小值。记住涌现网络必须要保证与原来的度序列相一致，所以和之前一样，链路对的调换仍然是有限制的。这种复杂组合算法需要多趟，每一个时间复杂度都是 $O(C_2^m)$。

确定性算法不断地形成链接对，从链路 1 开始，到链路 2，链路 3，……，链路 n，然后将链路 2 和链路 3 配对，……，一直到链路 $n-1$ 和链路 n 形成链路对。每个链路对都要检查排列是否保留了相同的度序列，并且没有创建重复或者是断开的网络。每趟要进行 $m(m-1)/2$ 次的比较，直到长度不再减少，否则仍然需要经历多趟。像图 7-11 研究的这种规模的网络大约需要 12 趟左右。

确定性算法需要花费数倍于 $O(C_2^m)$ 的时间步，这对于高的 m 来讲是非常大的。例如，1000 条链路的网络需要 $10(C_2^{1000}) = 4\ 995\ 000$ 时间步才可以完成。我们可以对这个随机涌现算法进行改进吗？结果证明，大多数情况是不能的。两种算法花费的时间几乎相同，因为节点是随机地分布在一个二维表面的。按照确定性的顺序做链路选择与通过随机排列选择一样地随机。实际上，主观感觉随机选择要花费更多时间去检查所有的链路对。正如我们下面将要看到的，实际上这两种方法没有什么差别。

7.6.3 最小长度涌现模型

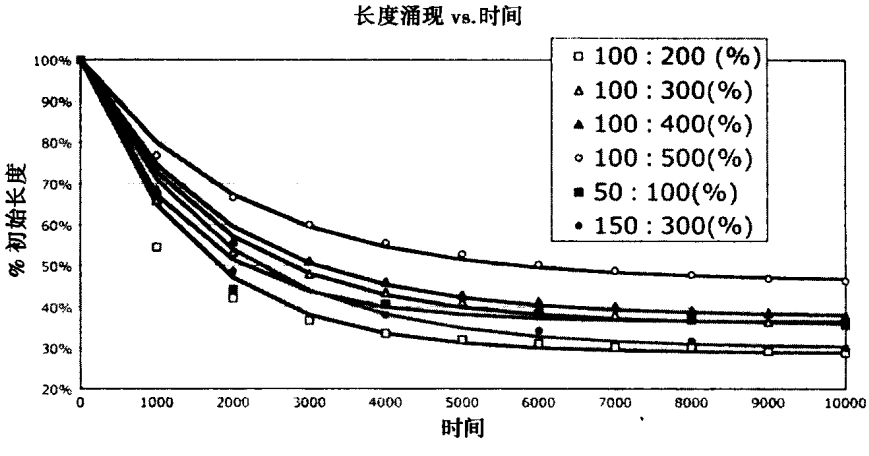


图 7-11 在时间步 [0, 10 000] 内，有 n 个节点、m 条链路的网络的最短长度/初始长度的涌现

图 7-11 说明了方法 NW_doShorter( )在一些随机生成和随机分散的网络上的执行结果，这些网络的大小范围是：$n = 50 \sim 400$，$m = 100 \sim 800$。很明显，最小长度网络是随着时间按照指数方式进行涌现的。图 7-11 绘制出了初始长度随时间变化的百分比，时间是从 $t = 0$ 到 $t = 10\ 000$。当 10 000 时间步不能够确保收敛到绝对最小值的时候，很明显涌现算法会在 10 000 时间步内无限接近于最小总长度，至少对图 7-11 中研究的那种大小的网络是这样的。

直观地讲，总长度的增长与 m 是成比例的，因为更高的 m 意味着更多链路加起来。总长度应该和 $1/\sqrt{n}$ 成反比地变化，因为 sqrt (n) 个节点是随机地分散到一个固定的区域。当 n 增长的

时候，我们必须要压缩节点之间的空间，以便让所有的节点都能够被包含在这个大小固定的区域内。因此，节点间的间隔距离以 $\sqrt{n}$ 的速率减小。

设 α 是固定因子，L 是总长度：

$$L \sim \alpha \frac{m}{\sqrt{n}} = \sum e_i$$

这里 e_i 是链路 i 的长度，求和是对于所有的链路进行的。

我们将最小总路径长度涌现看做是时间的函数。令 $L(t)$ 为时间步 t 时的总长度，$L_0 = L(0)$，$L_\infty = L(\infty)$ 为经过无限时间后确定的最小值。开始的时候，随机散布的节点被 m 条链路连接，m 条链路的总长度为 $L_0 = \sum e_i$，但是随着时间的推移，当前长度 $L(t)$ 和最小长度 L 的差别会越来越小。我们推断变化率与差值是成比例的，比例定义如上面所示：

$$\frac{\delta L}{\delta t} = \alpha \frac{m}{\sqrt{n}}[L_\infty - L(t)]$$

这个一阶微分方程的限制条件由 L_∞ 和 L_0 所指定，具有如下解：

$$L(t) = L_\infty + [L_\infty - L_0]\exp\left(-\alpha \frac{nt}{\sqrt{n}}\right)$$

经过除以 L_0 之后，将 $L(t)$ 转换成百分数形式的 $L'(t)$，然后用 $A = (L_\infty / L_0)$ 替代，我们得到图 7-11 中所描绘的实线的模型。这个模型含有参数 A 和 α：

$$L'(t) = \frac{L(t)}{L_0} = A + [1 - A]\exp\left(-\alpha \frac{mt}{\sqrt{n}}\right)$$

在图 7-11 中我们得到对 $A = L'(10\,000)$ 和 $\alpha = \alpha - \text{bar}$ 的估计，它们是在时间 t 上的平均值：

$$\alpha - \text{bar} = \text{average}\left\{\frac{-\sqrt{n}}{mt}\sum \ln\left[\frac{L'(t) - A}{1 - A}\right]\right\};\ t = 0,\ 1,\ 2\cdots,\ k$$

我们每 1000 个时间步抽样一次，以此来获取参数 k 的值，在图 7-11 中的抽样产生的 $k = 10$。一般来说，最小值 L_∞ 是 $O(m/\sqrt{n})$，但是正确的值取决于参数 α。α 越大，最小值 L_∞ 涌现的越快。

表 7-5 从图 7-11 获取的参数

n	m	$m\frac{m-1}{2}$	A	a	$a\frac{m-1}{\sqrt{n}}$
50	100	4 950	0.363	5.08E-05	7.19E-04
50	300	44 850	0.552	1.24E-05	5.28E-04
150	300	44 850	0.299	2.18E-05	5.34E-04
100	200	19 900	0.287	3.39E-05	6.78E-04
100	300	44 850	0.357	1.83E-05	5.50E-04
100	400	79 800	0.377	1.31E-05	5.26E-04
100	500	124 750	0.463	9.37E-06	4.68E-04

注：这里使用了“科学”记数法以节省空间（例如，5.08E-05 表示 5.08×10^{-5}）。

表 7-5 总结了一些有用的值和模型参数，这些值是通过图 7-11 所示的实验获得的。读者可以通过比较表 7-5 中的 A 列和最后一列中的值来验证最小值 L_∞ 与 $O(m\sqrt{n})$ 之间的关联。读者也可以验证在 $10[m((m-1)/2)]$ 时间步内可以找到最优解。

7.6.4 二维布局

有了上述这些理论，我们就可以将涌现方法应用到实际的布局问题中，比如印刷电路板上

的组件布线。假设我们将印刷电路板上一边的组件用线路点对点地连接到另外一边的组件上。这样，线路就没有必要围绕组件。我们想要通过最小化线路长度来节省线路的费用。

进一步来讲，组件是通过两条线路连接的——一条进，一条出。因此，度序列分布是所有的节点的度都是2的1－规则网络。实际上，我们通过生成一个1－规则网络来构造一个初始网络$G(0)$，对我们想要的组件（节点）在印刷电路板上进行布局（参见图7-12a）。一旦我们在印刷电路板上安排好了节点后，就应用涌现过程，并且在最小长度配置涌现时观察（如图7-12b所示）。

图7-12显示了初始网络布局，后跟通过涌现获取的最小长度结果。优化之前的线路总长度是1778个单位，在优化之后是1329个单位——减少了25%。同时要注意，最佳布局通过连接最近的邻居形成了一个环形图（环形网）。只要我们认识到这一点，就可以通过简单地将导线连接到最近的相邻节点，手工地连接任何1－规则网络。

最短长度的布线例子说明了一个总的思想：复杂的最优问题是无法简单地用方程或模型来描述的，这些方程和模型带有很复杂的限制条件，它们可以通过涌现来解决。例如，复杂流模型、严格度序列分布和非线性“费用模型”，就不可能通过分析来解决，但是通过涌现却很容易解决。然而，通过昂贵的最小长度涌现的分析，时间复杂度可能很高。

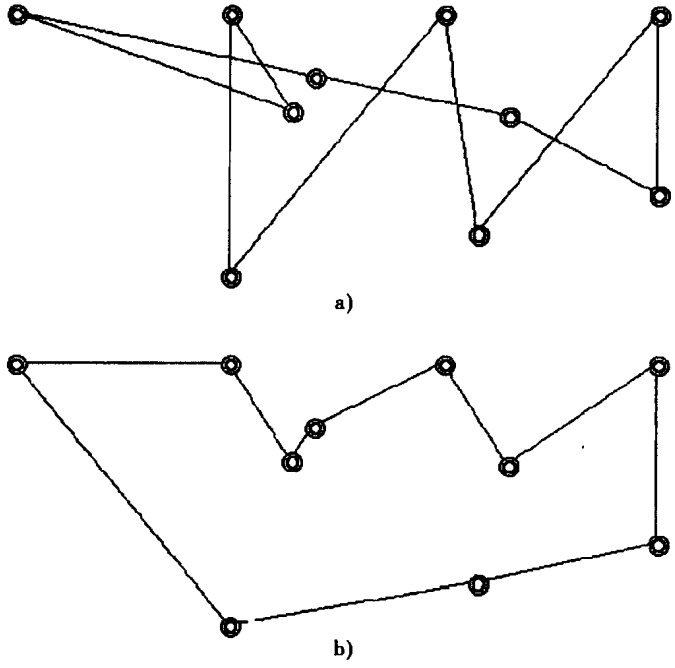


图7-12 在印刷电路板上的组件布局：a）优化前，$L_0 = 1778$的初始1－规则网络；b）经过随机涌现优化后，$L_\infty = 1329$的涌现1－规则网络

练习

7.1 包含$n = 100$个节点、$m = 500$条链路（图7-2）的“hub涌现网络”经过5000时间步后的最大hub度的近似值等于多少？经过100 000时间步后又如何？

7.2 当$n = 100$、$m = 1000$时，通过hub涌现需要经过多少时间步才能产生度为100的hub？

7.3 包含$n = 100$个节点、$m = 500$条链路（图7-3）的“hub涌现网络”经过5000时间步后的熵等于多少？经过100 000时间步后又如何？

7.4 使用程序Network.jar获得$I(G)$近似中的参数A、B、C、D的估计。对于$n = 100$、密度等于

5%的参数值等于多少？密度等于20%又如何呢？

7.5 如图7-3所示的密度为10%的hub涌现网络，经过$t = 10\,000$次重复后平均路径长度近似等于多少？

7.6 从Network.jar中收集结果，并对于图7-3的hub涌现网络构造一个平均路径长度与时间的模型。平均路径长度与最大hub度的关系模型是什么？

7.7 从Network.jar中收集结果，并为图7-3的hub涌现网络构造一个聚类系数与最大hub度的模型。聚类系数与逝去时间之间的关系如何？

7.8 对于$n = 100$、$m = 500$的随机网络，使用聚类系数涌现微规则，大约需要多少时间步才能将其聚类系数加倍？假设初始系数为0.10。

7.9 度序列涌现仅在允许重复链路时才能保证收敛。通过修改Network.jar中的NW_doAddLink()方法并在很多例子上运行“per node”（每节点）涌现算法证明该论断。

7.10 大约需要多少时间步，才能减少$n = 50$个随机散布的节点、$m = 250$条链路的随机网络的总的链路长度到达初始长度的75%？

7.11 考虑图7-13中的Koch字串网络模型。$G(0)$最初是由$n = 5$个节点、$m = 4$条长度等于1的链路组成，因此$G(0)$链路总长度为4。在每一时间步我们可以重复并行地应用如下规则到所有链路上：用$G(0)$的复制品替代G的每一条链路，但是链路长度是替代链路长度的$\frac{1}{3}$。如此一来，$G(1)$长度为$\frac{16}{3}$。那么$G(t)$的链路长度的表达式是什么，这种涌现过程收敛吗？

7.12 对于$n = 50$、$m = 200$的随机网络$G(0)$经过$t = 10\,000$次重复后，hub涌现对最大度导航路径长度的作用如何？

7.13 下列度分布中的哪一个是由$n = 5$的Mihail等人的涌现过程实现的？解释为什么不可实现序列是不能可实现的？

(a) {5, 4, 3, 2, 1}

(b) {2, 3, 1, 1, 1}

(c) {2, 3, 2, 2, 1}

(d) {1, 3, 2, 2, 1}

7.14 n个节点按照直径等于200的环形表面（盘子状）的1-规则网络拓扑布置的最短链路长度是多少？假定所有节点的度等于2，节点初始时随机放在圆形盘状的环形表面上。随着n的无限增长，最短总链路长度的限制值等于多少？

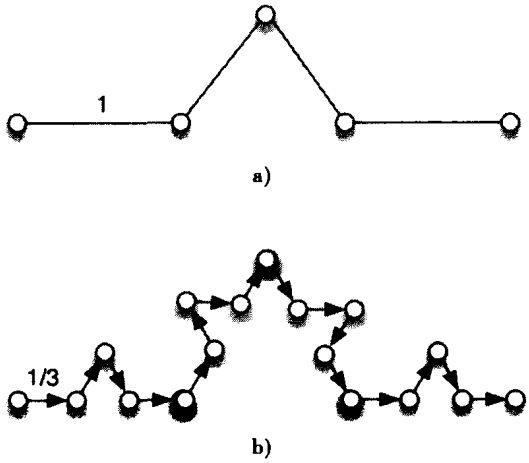


图7-13 Koch字串网络表示：a) $G(0)$；b) $G(1)$。在每一步串以$G(0)$的$\frac{1}{3}$复制品替代它的所有链路

传 染 病

传染病数学模型有着悠久的历史，最早可以追溯到1927年，当时Kermack和McKendrick首先推测出冠以其名的数学模型。Kermack-McKendrick是一个出生死亡模型——在某个人群中新的个体以一定的速度接触传染，已感染者要么死亡，要么以一定的比率康复，另外还假定人群的其他所有人是易受感染的。这个模型根据假设的复杂性，运用可以分析或数值求解的三种速率方程表示传染病传播的方式。

例如，SIR（susceptible-infected-removed）变量是指个体处于三个状态之一：易感、感染、消亡。个体以一定的概率从一个状态转换到另一个状态。一个易感个体以感染几率γ被感染，然后又以消亡几率Δ进入恢复或者消亡状态。SIS（susceptible-infected-susceptible）变量假定从传染病康复的个体仍然可能再次被感染。由于在某种条件下，对于持久（无限）传染的SIS来说，SIS模型特别有意义。

各种各样的更加复杂的模型都是由Kermack-McKendrick模型派生而来的。这些模型适用于阐述人类传染病领域的传播方式，同样也适用于其他各种领域。例如，传染病模型也被用来描述概念、产品以及物理系统中故障的传播。传染病模型在除了人类疾病传播外的其他一些领域也是十分有用的。

在20世纪90年代之前，大多数传染病模型假定均匀同质混合——易感个体的人群是均匀地通过一个地理区域传播，并且接触感染的几率对于所有个体都是均匀的。同质混合忽视了高度密集群的个体数、接近程度和接触频率。模型还假定每个个体最终会与所有其他个体接触。均匀接触简化了分析，但对于大多数传染病模型的应用来讲是不现实的。

本章是有关网络传染病的——考虑分析现实人口的非均匀结构的传染病。链路用于为接触建模，节点为个体建模。每个时刻个体可以处于一个状态：易感、感染、消亡或恢复。网络传染病是有关在整个网络上的接触传染。与经典的Kermack-McKendrick模型不同，网络传染病模型既要分析网络的拓扑，也要考虑感染率、死亡率、状态转换。

一般来讲，网络传染病是传染病通过网络广泛而快速传播的过程。典型的是，传染是一种由邻接节点通过沿着一条或多条链路传播而引起的网络节点状态（工作、故障、睡眠、激活等）。链路的传播也就是节点u通过网络链路被节点v传染：$u \sim v$。感染率$\gamma(v)$是指传染传播在节点v经过一条或多条链路成功感染邻近节点的概率。因此，传播是关于节点状态、网络拓扑、感染率的函数。

网络传染病为支持信号传播的社会和物理系统的研究创造了极好的模型：

1. 生物组织的疾病传播，例如人群中常见的感冒。
2. 经过互联网的恶意软件（蠕虫，病毒）传播。
3. 对产品及音乐、电影、衣服样式等时尚兴趣的传播。
4. 思想、概念、新闻、流行观念（例如地球是宇宙的中心）的传播。
5. 材料、系统的故障或者错误的传播，例如电网中断的多米诺效应。

网络传染病是一种与网络的涌现和稳定（将在下一章讨论）相关的动态过程。从概念上来

说，传染传播导致系统的不稳定（传染）非常像在非线性系统中的混沌。如果这种传染消亡了，那么系统将达到一个稳定点并保持下去。如果传染复发的话，那么系统将会（无限地）摆动。在传染病学中，消亡了的传染病都是 SIR 过程的成员，那些没消亡的传染病是 SIS 过程的成员。一个在易感个体和感染个体之间无限振动的 SIS 过程也称为持续过程。

本章开始研究 SIR 和 SIS 过程，如下所示：

1. 网络传染病是一个节点状态或者节点条件经链路通过网络传播的过程。网络传染病不同于传统的传染病，因为传播方式是由网络拓扑和传统参数如感染率 γ 、恢复率 Δ 和恢复时间 t_r 决定的。

2. 有两个一般的传染病分类：SIR（易感 - 感染 - 消亡）和 SIS（易感 - 感染 - 易感）。SIR 网络传染病最终消亡了（要么每个个体感染被消除，要么没有个体继续被感染），而 SIS 网络传染病根据阈值 τ 决定消亡与否。

3. 传染病阈值 τ 确定 SIS 传染病是持续存在（$\tau > 1$）还是消亡（$\tau < 1$）。在传统的 Kermack-McKendrick 传染病模型中，它假定大的均质的人群，τ 是 γ 和 Δ 的函数，但对于有限的非均质的网络，我们说明了 τ 大多数情况下是由网络谱半径 ρ 来决定。

4. 一个 SIR 网络传染病的峰值传染密度服从平均度为 λ 的幂律函数：$i_{max} = A((1-B)\lambda^q)$，这里参数 A 、B 和 q 由曲线拟合来确定。这种近似法适用于随机、小世界、无标度网络，但不同的网络采用不同的参数值 A 、B 、q 。

5. 经过足够时间后，基于一个随机网络的 SIS 传染病既可以概率 τ 消亡，也可以概率 $(1-\tau)$ 成长到固定点 $i_\infty = (1-\tau)$，这里阈值 $\tau = \Delta/v$，$v = \gamma\lambda$ 。因此，有限的随机网络的阈值和固定点密度是由感染率 γ 、恢复率 Δ 和网络连通性 λ 所决定的[⊖]。

6. Z. Wang、Chakrabarti、C. Wang 和 Faloutsos 的 WCWF 模型对任何谱半径为 $\rho(A)$ 的网络定义阈值为 $\tau = \rho(A)$[⊜]。我们得到一个改进的二次方程 WCWF 模型：$\tau = \Delta/\gamma = \rho(A)(B - A\gamma)$，这个模型改善了 WCWF 对更大概率范围的近似。其中参数：

$$A = O\left(\lambda + \frac{\lambda}{\rho(A)}\right), B = O\left(\frac{\lambda}{\rho(A)}\right)$$

7. 我们证明了固定点感染密度 i_∞ 线性地随任意网络的谱半径而减少，按照 $i_\infty = b - m\rho$ 接近阈值。对于这里研究的实验性网络（随机、无标度、小世界和星形网络，$n = 100$，200 个节点），$b = 36\%$ 和 $m = 1\%$ 。

8. 抗原是对付像传染病一样传播传染性的传染对策，但它要从传染节点清除感染并为易感节点接种，使之将来不再受到感染。我们说明最有效的抗原对策部署是从 hub 或最大紧度节点处开始。最小效率的抗原对策部署是从任意选择的节点处开始。

9. 我们认为“治疗互联网”和类似网络的最好办法就是策略性地在网络 hub 或最大紧度的路由器和交换机上接种抗原。最小效率方法是在桌面和便携式客户机计算机上部署抗病毒软件，就像我们当前所做的。

本章使用 Network. jar 程序去生成和分析这里描述的传播病。本章中所介绍的 Java 方法都是在 Network. jar 中找到的代码的简化版本，并且由 Network. jar 前面板上的 INFECT 和 ANTIGEN 按钮所激活。

8.1 传染病模型

传染病的数学建模追溯到 1927 年的传统 Kermack-McKendrick 方程的公式化表示。这种开创

⊖ 连通性和平均度在随机网络中是同一概念。

⊜ WCWF 定义 $\tau = \gamma/\Delta$，因此其结果是 $\tau = 1/\rho(A)$ 。

性的模型进行了很多简化假设，虽然过去几十年人们不太关注这些假设，但它仍用做基本的出发点。我们首先对它加以描述，然后会指出假设不再应用于有限网络。接下来，我们针对网络调整了传统的模型，这种模型创立了网络传染病学的基础。

Kermack-McKendrick 模型提出了同质混合——人群中的个体都是一致地隔离并且都是一致地彼此互相暴露的。这种模型还提出了这种人群是十分巨大的，大到足以忽略 n 的有限性。一般情况下这种理论是起作用的，只是当应用于有限网络时其作用是有限的。人群的有限性增大了网络中传染病精确建模的难度，因为网络拓扑不能再被忽略。进一步来讲，假定个体与总人群中较小的子集接触更加现实，因此网络能更准确地反映真实生活。

我们给出传统模型并进一步改造将其适用于有限的结构化网络上。本章回答的主要问题之一是："去掉同质混合假设后会对网络模型产生什么影响?" 这里要解决的另外一个有关结构的问题是："网络拓扑在决定网络传染病传播中扮演什么角色?" 我们证明拓扑主要决定通过它的*邻接矩阵的谱半径*测量的网络中的传染病的传播。

8.1.1 Kermack-McKendrick 模型

传染病的第一个数学模型是由 W. O. Kermack 和 A. G. McKendrick 于 1927 提出，并且以他们的名字命名。Kermack-McKendrick 模型假定同质混合（每个人都以均等的机会从其他人那里感染疾病）以及大的人群（这里 n 是无限的）。

在接下来的叙述里，将交替使用术语个体和节点，以便与社会网络分析文献保持一致。个体的*连通性*就是它的度，*邻接节点*就是它的邻居。网络连通性由平均度 λ 来描述。

Kermack-McKendrick 模型根据以下 4 种状态之一将每个个体分类：（1）如果一个个体可能被感染，那么它就是*易感的*；（2）如果个体已经接触到了传染源，则它就是*感染的*；（3）如果个体从感染中恢复并且对以后的感染免疫时，则它就是*恢复的*；（4）如果一个个体由于传染而死亡的话，那么它就是*消亡的*。我们将 γ 定义为感染率，死亡率或恢复率定义为 Δ，一个节点在转换为消亡或者恢复状态之前感染状态持续时间定义为 t_r。

定义易感个体数为 $S(t)$，感染个体数 $I(t)$，人群中删除掉的最终死亡数或从疾病中康复的免疫个体数为 $R(t)$，时间为 t。在有限的人群中，$n = S(t) + I(t) + R(t)$。Kermack-McKendrick 方程将 S、I 和 R 相互之间通过时间变化率和初始状态联系起来，如下：

$$\frac{\delta S(t)}{\delta t} = -\gamma S(t)I(t); S(0) = S_0 \tag{I}$$

$$\frac{\delta I(t)}{\delta t} = \gamma S(t)I(t) - \Delta I(t); I(0) = I_0 \tag{II}$$

$$\frac{\delta R(t)}{\delta t} = \Delta I(t); R(0) = R_0 \tag{III}$$

$$S(t) + I(t) + R(t) = n \tag{IV}$$

等式（I）中说明了易感个体数量的变化率随着易感个体和感染个体的数量上升而下降。这同样也受限于等式（II），因为易感个体的数量以与已经感染个体数量相反的方向增加或减少。等式（II）说明了感染个体数量的变化率根据易感个体和感染个体的数量的比例增加或减少，其中还需要减去因感染而消亡个体的数量。等式（II）的逻辑在于感染从感染个体传播到易感个体。易感者和已经感染者越多，传染病扩散的就越快。等式（III）表达了已经感染个体数和它们消亡速率的一个简单的线性关系。等式（IV）说明人群保持了恒定而与每个状态中的数量无关。

使用标准化版本的 S、I 和 R 即所谓的"密度"加以替换，这样研究更容易进行：

$$s = \frac{S(t)}{n}\ i = \frac{I(t)}{n}\ r = \frac{R(t)}{n}$$

在这个例子中，我们使用小写的注释符号，而不更改 γ 和 Δ。Kermack-McKendrick 等式尽管仍然不变，但是 s、i、r 的解落在了区间 [0, 1] 而非 $[0,n]$ 内。按照标准化或每个状态下个体密度的名义，我们可以得到：

$$\frac{\delta s}{\delta t} = -\gamma si; s(0) = s_0$$

$$\frac{\delta i}{\delta t} = \gamma si - \Delta i; i(0) = i_0$$

$$\frac{\delta i}{\delta t} = \Delta i; r(0) = r_0$$

$$s + i + r = 1$$

例如，假设 $\Delta = 0$，$s + i = 1$，并且求解 $i(t)$：

$$\frac{\delta i}{\delta t} = \gamma si = \gamma (1 - i) i; i(0) = i_0$$

在差分方程中重新布置各项，并进行部分积分：

$$\frac{\delta i}{i(1 - i)} = \gamma dt$$

$$\ln(i) - \ln(i_0) - \ln(1 - i) + \ln(1 - i_0) = \gamma t$$

ln ( ) 表示的是自然对数。让 $A = (1 - i_0)/i_0$，求解 $i(t)$：

$$i(t) = \frac{1}{1 + A\exp(-\gamma t)}; t \geqslant 0$$

这就是在很多学科中都可以发现的，用于处理增长现象的著名的*逻辑斯蒂增长曲线*。它的几何形状很像“S”，因此也可以称为 S 曲线。在假设 $\Delta = 0$ 的条件下，简化了的 Kermack-McKendrick 方程解，服从逻辑斯蒂增长曲线，像传染病一样从某一小值增长到 100% 的人群，从而描绘得出 S 形逻辑斯蒂增长曲线（如图 8-1 所示）。

我们继续处理例子，假设 $i_0 = 0.05$（5%），$\gamma = 0.75$（75%），在时间 $t = 10$ 时感染个体的密度为 98.96%：

$$A = \frac{1 - 0.05}{0.05} = 19$$

$$i(10) = \frac{1}{1 + 19\exp(-0.75(10))} = 0.9896\ (98.96\%)$$

$i(t)$ 与 t 之间的关系如图 8-1 所示。当人群数 $n = 100$ 时，那么 99 位个体将会在 $t = 10$ 时被感染。

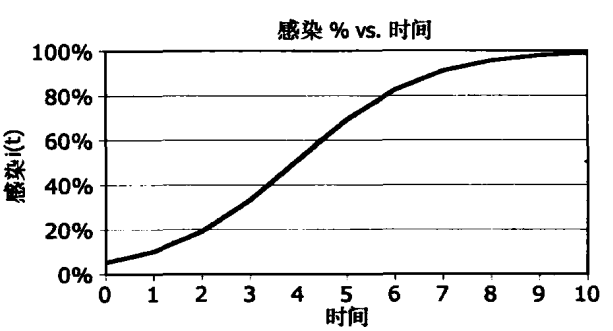


图 8-1 $\gamma = 75\%$ 和 $i_0 = 5\%$ 时的逻辑斯蒂增长曲线

8.1.2 传染病阈值

非线性效应，例如人口达到传染病阈值并保持下去，或者在感染和易感两个状态之间振荡，

都会使得传染病的表现变得复杂，特别是在一个有限的人群里更是如此。在某些传染病中，传染是永远不会灭绝的，而其他某些传染病，则在感染个体数量到达峰值后下降。传染病阈值 τ 是决定一个传染是否无限反复发作或者最终灭绝的感染概率。在某一特定的条件下，当感染率 $\gamma < \tau$ 时，一个传染病将会消亡；否则，将持续下去。当 $\gamma \geqslant \tau$ 时，我们说传染病持续存在并达到一个固定点（$0 < i_\infty < 1$），并且会无限地保持下去。传染病阈值 τ 决定了感染率和恢复率/死亡率之间的平衡，这种平衡勉强支撑一个持续存在下去的传染病，i_∞ 就是一个持续存在的传染病的固定点值。当传染病是持续稳定的，那么就是局部流行的；如果广泛传播的话，则是大规模流行的。

在何种条件下传染病会变得持续稳定，并达到一个不会消亡的传染水平，我们有兴趣研究这个问题。这将导致对下列问题求解：

1. 将持续存在下去的传染病与消失的传染病区别开来的阈值 τ 是多少？
2. λ 和 Δ 在什么条件下会导致持续稳定的传染病？
3. 持续稳定的传染病的固定点 i_∞ 是多少？
4. 网络传染病与传统的传染病是否不同？如果不同，为什么？
5. 针对传染病的最有效的对策有哪些？

前两个问题通过精心为基于阈值的感染率和消亡率建模来求解，之后修改传统 Kermack-McKendrick 方程以便精确地为网络传染病建模。我们说明了为了获得精确结果必须考虑网络拓扑。第三个问题是通过曲线拟合仿真数据以便画出固定点密度与谱半径 ρ 来回答。我们说明在整个研究的网络范围内它们的关系是线性的。

我们也解释了为什么网络传染病不同于传统传染病模型，网络传染病是由网络拓扑确定的。与网络是随机、无标度或小世界无关，只跟通过平均度和谱半径测量的连通性有关。

最后，我们证明根除病毒传染病的最佳对策，比如基于互联网的计算机病毒的传播，是一个始于网络 hub 的抗原传染蔓延。hub 接种会导致更快的根除和更低的峰值感染。这个重要结果对于在互联网上与恶意软件斗争有着直接的意义。

8.1.3 易感－感染－消亡（SIR）模型

SIR 模型在动物群的研究中具有特别意义，因为它是生物的实际建模：某个群体中的每个个体一般会经历表示传染过程的几个状态：易感，感染以及康复或消亡。“易感”是说某个个体容易受到感染，“感染”是说个体已经感染了，“康复”是说从感染中恢复过来，而“消亡”是说个体死去而不能再被感染或者康复。

SIR 模型被严格定义成状态图（参见图 8-2a）。初始时，所有的节点都是易感的，因此这种方式可以应用于群体中的每个个体。然后群体的小部分 i_0 接触感染源并传染给它们的邻居，并在这个群体间传播。

在图 8-2 中，“tr”是指 SIR 模型中的个体恢复时间 t_r，在感染个体消亡或者恢复之前它会持续感染“tr”个时间单位。感染个体按照死亡率或恢复率的概率消亡，并对随后的感染以概率等于“1－死亡率”进行免疫。

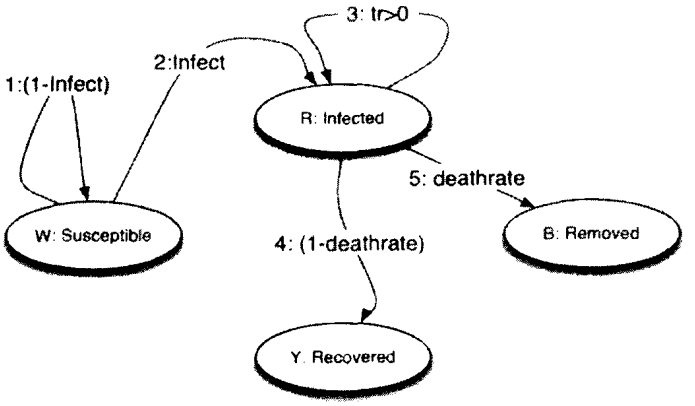
初始时，群体中感染了很小的百分比。但是，如前面所说，感染个体的比率是按照 S 形曲线增加的，直至到达一个最大的感染峰值。之后，感染个体的密度随着已感染、恢复或者消亡的数量到一个最大值而减少——显然易感个体的数量更少了。已感染个体的密度必须减少，由于死去或免疫的个体密度增大了。如此一来，SIR 传染病轮廓就像波浪形一样（参见图 8-3）。

与 SIR 模型相反，SIS 模型允许群体中个体恢复后再次对传染易感。在自然界和互联网上可观察到反复发作或持久的 SIS 传染。在下一节中，我们将回到 SIS 模型的讨论之中。

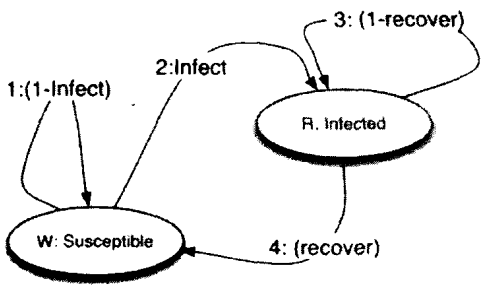
图 8-2 的状态图诠释了每种状态变化的概率，在每种迁移链路上都带有一个标签，除了 SIR 状态图中的迁移 3 之外。例如，“2：infect”表示迁移 2 以概率 γ 发生。对于 SIR 状态图，“3：

tr>0”表示只要感染持续大于0，一个感染个体就仍然处于感染。对于每个个体，在个体感染后就设置 t_r，在每个时间步减1，直到零为止。

节点标上颜色：白色指示易感状态，红色指示感染状态，黑色指示删除消亡状态，黄色指示康复状态[⊖]。一旦一个节点删除（黑色）或康复（黄色），它就处于删除或康复状态。



a)



b)

图 8-2 传染病模型：a）SIR；b）SIS（仿真程序 Network. jar 使用的颜色代码：W－易感的－白色；R－感染的－红色；B－消亡的－黑色；Y－康复的－黄色）

SIR 模型近似于自然界发生的情况，某个个体感染后按照几率 γ 传染传给周围的个体。每次感染会持续 t_r 个时间单位。当度过康复/消亡时间后，个体要么以几率 Δ 被感染致死，要么以概率（$1-\Delta$）康复。

感染密度曲线的精确图形是由 γ、Δ 和 t_r 来确定的。很显然，处于波峰值的感染人群密度越高，感染率 γ 就越高（参见图 8-3a）。但是，当感染率更低时，因为传播慢下来，波的时间跨度就被延伸了。图 8-3 表示感染率在传染波峰值如何影响感染个体的最大密度，以及感染活跃传播的时间长度。

提高消亡率 Δ 会增加人群中消亡或康复的人群百分比，但是对感染百分比几乎没什么影响。它仅改变从消亡到康复个体的比率。图 8-3c 中，当 Δ 越高，死亡率越高；Δ 越低，死亡率则越低。这具有一定意义，因为 SIR 只有当个体被感染过后才除去个体，并且易感个体百分比是一样的，而与死亡个体的百分比无关。

总而言之，增加恢复时间 t_r 也会提高感染个体的百分比，使得波形曲线变得更宽（参见图 8-3b）。对于给定感染率，更长的恢复时间会使感染波形变平并提高峰值。从直观上来讲，这是因

⊖ 这些颜色对应于 Network. jar 中节点的颜色方案。

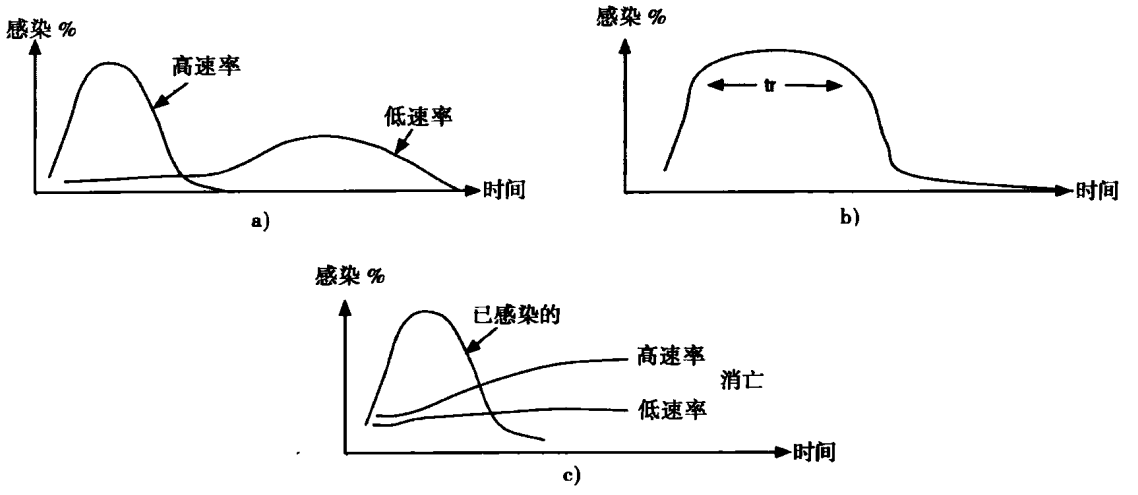


图 8-3 感染个体密度与时间（依据不同传染病系数值）：a）感染率变化；b）长的康复时间；c）死亡率变化

为感染个体在以几率 Δ 康复或死亡之前保持感染状态更长的时间。个体感染的时间越长，就越有机会感染给其他个体。

随机网络中的 SIR 模型的感染波峰值也会随着随机网络连通性的提高而增加。回顾一下，连通性是另一种表示随机网络密度的方法，因为当平均度增加时，密度也在增加。如果我们保持 $\gamma\Delta$ 和 t_r 恒定，那么随机网路中的 SIR 感染峰值必须按照函数 λ 而增加。

这刚好就是我们在图 8-4 中对于 $\gamma = \Delta = 20\%$ 和 $t_r = 10$ 时所观察到的。对于随机网路，峰值感染密度服从幂律分布：

$$i_{\max} = A\frac{1-B}{\lambda^q}$$

这里 A、B、q 由曲线拟合来确定。图 8-4 中所示的完美拟合是从参数 $A = 1.03$、$B = 4.0$、$q = 2.4$获得的。除了感染率、恢复率、感染持续时间，图 8-4 中显示峰值感染也同样与网络的连通性有关。不仅如此，我们能够用幂律函数估计随机网络的峰值感染。这个实验是有局限性的，但是它以一种重要的方式——与网络拓扑有关，证明了网络传染病区别于传统 Kermack-McKendrick 传染病。

那么对于非随机网络，峰值感染密度和连通性之间的幂律关系是否仍然成立呢？当 SIR 传染病暴发时，无标度和小世界网络的峰值感染率又如何呢？

8.1.4 结构化网络峰值感染密度

图 8-4 中显示峰值感染密度是连通性的函数，因此也是随机网络中密度的函数。那么对于像小世界或无标度群体的结构化网络是否也是一样的呢？答案是肯定的，但网络类型不同，相应的幂律分布也有所不同。对于随机、小世界和无标度网络，峰值感染密度的不同是由于幂律指数不同所造成的（参见表 8-1）。连通性越高，相应的每类网络的峰值感染密度就越高。但是，这种增长率不是直接增长的，小世界网络位于参数值谱范围的一个极端，无标度网络则位于另一极端。我们将说明：感染传播的速率和程度范围估计必须集成密度和结构化度量，如谱半径。

表 8-1 中显示幂律指数最大的是无标度网络，其次是随机网络，小得多的是小世界网络。很容易理解为什么无标度网络有这么高的峰值，因为 hub 中心节点是传染的超级传播者。具有大量 hub 的网络非常具有传染性。为什么随机网络几乎具有相似的传染性就不那么明显。小世界网络

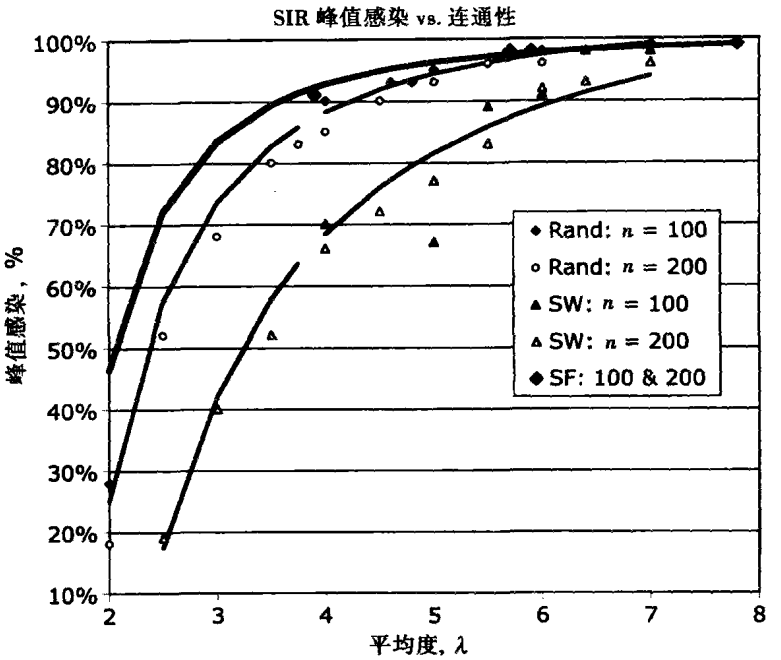


图 8-4 随机、无标度及小世界网络的峰值感染密度与连通性（平均度）对比，其中 $\gamma = \Delta = 20\%$，$t_r = 10$。数据是针对 Erdos-Renyi 随机网络、Watts-Strogatz 小世界网络（$p = 10\%$）和 Barabasi-Albert 网络（$\Delta m = 2$），分别针对 $n = 100$ 和 $n = 200$ 个节点采集的

是最不具传播性的，因为它们有聚类而不是 hub。聚类在紧密联系的邻居间传播要比远距离的更容易。当链路密度降低时，这种影响变得更加明显，如图 8-4 所示。同样可以从表 8-1 中看到相对较低的指数，$q = 1.7$。

图 8-4 证明了重要的一点：感染密度不仅仅是由平均度 λ 测量的网络密度的函数。很显然，无标度网络的平均度既不能代表其 hub 的连通性，也不表示平均度与小世界聚类相关。什么度量能够将网络密度和拓扑结构集成到一起呢？回顾谱半径 ρ 是任意网络中测量密度和违背随机网络结构的量。如果我们使用谱半径而不是平均度来表示网络的连通性，结果会如何呢？

图 8-5 画出了随机、小世界和无标度网络的谱半径与连通性 λ 的对比关系图。我们从第 2 章知道，随机网络的谱半径和平均度 λ 是等同的。星形网络的谱半径为 $\sqrt{(\text{hub})}$，其他所有类型的网络都落入其间。当拓扑不是随机的时，而是位于纯随机和纯结构之间，谱半径不仅测量连通性，同样也会测量拓扑的不均匀性。与平均度或同质混合相比，谱半径是传染病建模的一个更好的属性。

表 8-1 图 8-4 参数的曲线拟合值

参数	网络类		
	随机	小世界	无标度
A	1.03	1.10	1.00
B	4.00	4.00	4.00
q	2.4	1.7	2.9

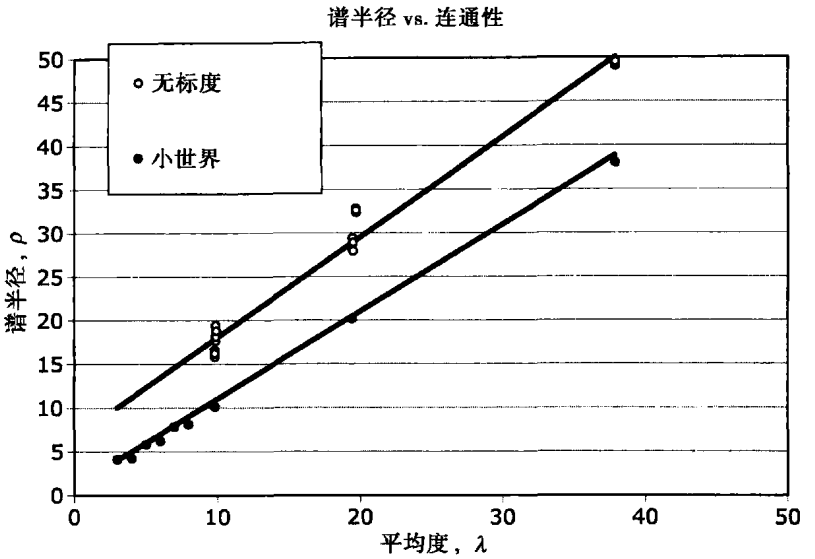


图 8-5 $n = 100$，200 的无标度和小世界网络的谱半径 ρ 和平均度 λ 之间的线性关系

图 8-5 显示谱半径 ρ 随着 λ 线性变化，但都是根据不同的网络类型有着不同的斜率 $\rho(\lambda) = A\lambda + B$，这里 A 和 B 分别代表斜率和截距。

我们通过对图 8-5 中提供的数据进行直线拟合得到 A 和 B 值。对于小世界网络，我们假设 10% 的重联概率，并获得下面参数值：

随机：$A = 1$，$B = 0$。

小世界：$A = 1$，$B = 1$。

无标度：$A = 1.15$，$B = 6.5$。

无标度网络的谱半径要比平均度高，并以平均度函数更快地增加（比起随机和小世界网络）。这就说明与平均度相比，谱半径能够对网络结构和连通性进行更好的测量。事实上，我们用这种方法解释 SIS 传染病中的传染病阈值。

假设小世界网络和无标度网络每个都有平均度 $\lambda = 20$。使用上面的近似值公式可以得到，小世界网络的谱半径是 ρ（小世界）$= (1)(20) + 1 = 21$。无标度网络的谱半径是 ρ（无标度）$= (1.15)(20) + 6.5 = 29.5$——要比小世界的谱半径高得多。由此可以证明，网络的谱半径越高，就越可能存在持续性的传染病。

8.1.5 易感－感染－易感（SIS）传染病

图 8-2b 显示了一种 SIS 传染病中的个体状态图。个体一般处于两个状态之一，即感染或易感。个体一般开始处于易感状态，然后每一个时间步后以概率 γ 迁移到感染状态。然后，要么以 Δ 恢复到易感状态，要么以 $(1 - \Delta)$ 保持感染状态。个体在易感和感染中不断循环，之后回到易感状态，因此得名 SIS。最终，个体群要么完全恢复，达到一个感染个体相对稳定的稳定状态，要么在感染数的高、低值之间波浪式地振荡。

SIS 传染病除了发生在生物学中外，还发生在其他很多学科之中，它为思想、时尚潮流和产品的传播建模，因此得到了市场研究人员的青睐。它们可以为错误的传播，以及像电力网络、互联网、交通系统等的物理系统的波浪式拥塞的传播建模。因此，对于理解传染病可能消亡、振荡或持续存在十分重要。永不中止的传染病的可能性会导致我们问三个关键问题：

1. 在什么条件下，才能保证传染病一定消亡呢？
2. 如果传染病永不消亡，那么期望的感染等级是多少呢？

3. SIS 网络传染病与生物传染病有区别吗？如果有，区别在哪里，为什么？

在开始之前，假定我们针对一个非常简单的、从不消亡的 SIS 传染病求解 Kermack-McKendrick 方程。因为非常明显的原因，我们将此传染病称为持续稳定的传染病。我们证明对于某种条件下的 γ 和 Δ，Kermack-McKendrick 模型的一个简单的版本形成了一个稳定传染病。

我们采用如下策略：

1. 通过模仿简单的生死过程，形成一个特殊版本的 Kermack-McKendrick 方程，其中出生（感染）率为 γ，死亡（恢复）率为 Δ。

2. 注意，在固定点的感染密度 i_∞ 下，$\delta i/\delta t = 0$。通过设置 $\delta i/\delta t = 0$ 求解固定点感染密度。

3. 注意在什么条件下产生固定点值的解。对于 SIS 传染病来说，这个条件是传染病的阈值。

让固定点感染百分比 i_∞ 为在 t 接近无穷时 $i(t)$ 的稳定状态值[⊖]。因为个体要么已被感染，要么是易感的，$s(t) + i(t) = 1$。进一步来讲，γ 为感染率，Δ 为个体恢复率，导致对于这种特殊情况下的 Kermack-McKendrick 密度方程如下：

$$\frac{\delta i}{\delta t} = \gamma st - \Delta i = \gamma(1 - i)i - \Delta i = [\gamma(1 - i) - \Delta]i$$

随着时间无限制地增加，$i(t)$ 变化率的阶数下降到 0，于是：

$$\frac{\delta i}{\delta t} = 0 = [\gamma(1 - i_\infty) - \Delta]i_\infty$$

求解 i_∞ 就得出持续稳定的传染病的百分比为：

$$i_\infty = 1 - \frac{\Delta}{\gamma}; \text{假设 } \Delta \leqslant \gamma$$

$$i_\infty = 1 - \tau, \text{这里 } \tau = \frac{\Delta}{\gamma}; \text{对于扩散的传染病则 } \tau < 1$$

例如，假设 $\Delta = 15\%$，$\gamma = 25\%$，则 $\tau = 15/25 = 60\%$。感染人群开始于 $i_0 < i_\infty$，然后升高到 $i_\infty = 1 - 60\% = 40\%$，这时保持稳定。SIS 传染病不是升到峰值然后重新降到 0，而是 SIS 传染病可能稳定地升到了一个特定值，然后就永远保持下去。

反过来，在这个模型的十分有限的限制条件下，对于 $\gamma < \Delta$ 的值 SIS 传染病消亡，因为 $i_\infty \leqslant 0$。在这个模型中，感染率一定至少要等于恢复率，才能维持持久的传染病。否则，SIS 传染病最终到达它的稳定状态上限 i_∞。

这个简单的模型回答了上述有关极端简化的 SIS 传染病的前两个问题。尽管如此，这个模型却没有集成网络结构对传播速率、阈值、稳定状态的影响。因此，这不足以描述 SIS 传染病是如何在任意网络中传播的。

8.2 网络中持续稳定的传染病

网络传染病模型在几个重要的方式上与传统的 Kermack-McKendrick 模型不同。首先，我们不能假设人群的均匀性——大多数的有实际意义的网络有着非均匀性的结构，例如，小世界或无标度网络便是如此。更具意义的是，我们已经证明了网络中的感染传播是极大地受平均度 λ 影响的，具体要根据网络的拓扑而异。网络连通性问题，因为每个节点是暴露在 λ 个邻接节点，而不是整个群体 n[⊜]。传染传播到邻近节点受限于总的感染率 ν，它倾向于接近 $\nu = \gamma/\lambda$，而非 γ。

由于网络拓扑控制着网络传染病的传播，我们就不能像在 Kermack-McKendrick 模型中那样

⊖ 这种固定点也被某些作者称为局部性平衡，并被标记为 i_0。

⊜ 在本书中交替使用平均度和连通性。

假设均匀混合。相反，网络传染病是由特定网络的度序列分布决定的，并且可以用网络的谱半径来描述。

另外，网络是有限的，因此我们不能用平均场理论来近似估计（小）网络传染病的行为。Kermack-McKendrick 模型的决定性模型和平均出非均匀网络结构的平均场理论模型不能精确地描述传染病的行为。相反，我们必须借助于统计学模型或概率论模型或精确表示非均匀性的拓扑的模型。

在下一节中，我们将会回答在本章开头提出的几个基本问题：

在什么情况下，SIS 传染病会自我存在下去？

何谓阈值？

什么是稳定状态固定点感染密度？

网络传染病模型是如何区别于均匀的、无穷大人群模型的？

我们证明将网络连通性 λ 和谱半径 ρ 结合起来可以提供对随机、小世界和无标度网络的足够描述，以便于我们可以从这两个参数精确地推测出任意网络的阈值和固定点感染密度。

8.2.1 随机网络传染病阈值

假设有 n 个节点、连通性由平均度 λ 给定的随机网络。进一步假设感染节点的密度由感染率 γ 和平均度 λ 所控制。设 $\nu = \gamma\lambda$ 是有效的传播速率，重写 Kermack-McKendrick 方程如下：

$$\frac{\delta i}{\delta t} = \nu i(1-i) - \Delta i; i(0) = i_0; \nu = \gamma\lambda$$

这个方程与标准的 Kermack-McKendrick 方程是一样的，除了将 γ 替代成 ν 之外。我们知道 ν 比 γ 能够更好地表示传染病的传播，因为每个节点都只直接与它的 λ 个邻近节点交互，而非留下的 $(n-1)$ 个节点。这个由 Kephart 和 White（KW）在 1991 年提出的模型，近似估计在随机网络中的持续稳定的 SIS 传染病，在阈值 $\tau = \Delta/\nu < 1$ 时，得出固定点值 $i_\infty = (1-\tau)$（Kephart，1991）。

我们按照 Kephart 和 White 的固定点和阈值推导，然后增加新的分析，以解释在仿真的网络传染病中所观察到的振荡。这些振荡通过数字求解连续变量 KW 解的离散差分方程模拟得以解释。这个分析策略如下：

1. 可以利用 Kermack-McKendrick 方程的次变化（这种变化通过逻辑斯蒂增长曲线可以得到），求解 KW 方程。

2. 求解离散差分方程模拟以便显示振动来自哪里。

3. 利用这些限制可以找到传染病阈值和固定点。

Kermack-McKendrick 方程的 KW 修改版对 $i(t)$ 的求解像以前一样形成一个逻辑斯蒂增长曲线。唯一不同的是 γ 被 ν 代替了：

$$\frac{\delta i}{\delta t} = \nu i(1-i) - \Delta i = i[(\nu - \Delta) - \nu i]$$

两边都除以 $i[(\nu-\Delta)-\nu i]$，我们得到：

$$\frac{\delta i}{i[(\nu-\Delta)-\nu i]} = \delta t$$

现在左边从 i_0 到 i 进行积分，右边从 0 到 t 进行积分：

$$\ln\left(\frac{(\nu-\Delta)-\nu i}{i}\right) - \ln\left(\frac{(\nu-\Delta)-\nu i_0}{i_0}\right) = -(\nu-\Delta)t$$

对两边应用 exp() 函数，将感染阈值 $\tau = \Delta/\nu$ 代入，调整后得出关系：

$$i(t) = \frac{i_0(1-\tau)}{i_0 + (1-\tau-i_0)\exp(-v(1-\tau)t)}$$

注意当随着时间无限地增加时，exp() 会接近于0，并将 i_0 从分子和分母中消去，因此固定点解 i_∞ 可以很容易作为阈值 τ 的函数得到：

$$i_\infty = 1-\tau, \text{当 } \tau = \frac{\Delta}{v} \text{ 时；或者定义 } \tau = \frac{v}{\Delta} \text{ 时}^{\ominus}, i_\infty = \left(1 - \frac{1}{\tau}\right)$$

固定点的解在阈值之下或 $\tau < 1$，因此这个条件保证一个持续感染。但是这个确定性的解忽视了在实际中所观察到的密度的统计学波动（参见图8-7）。SIS 传染病消亡的概率，由于随着剧烈波动的增加而增加，最终达到零。

这些波动的来源可能是什么呢？有两个基本的原因：（1）感染的传播是随机的——感染传播是概率性的（确定性模型忽略了统计学的变化）；（2）确定性的方程对于某一输入参数可能是不稳定的——对某些感染率和网络连通性 λ，解可能变成混沌。在大多数情况下，混沌振荡无法与随机变异区别开来。

通过考虑 KW 差分方程的离散模拟，我们可以得到一个对 KW 模型的混沌而不稳定的表面而直观的理解。

离散模拟是通过用离散差分 $i(t) - i(t-1)$ 替换连续导数 $\delta i/\delta t$ 得到的有限差分方程：

$$i(t) - i(t-1) = i(t-1)[v(1-i(t-1)) - \Delta i(t-1)]$$

将 $i(i-1)$ 换到右边产生了一个迭代形式：

$$i(t) = i(t-1) + i(t-1)[v(1-i(t-1)) - \Delta i(t-1)]; t = 1,2,\cdots$$

$$i(0) = i_0$$

已知初始值 $i(0)$，我们可以通过时间迭代得到针对这个有限差分方程的一个数值解。绘制有限 $i(t)$ 与 t 之间的图，其中 $\gamma = \Delta = 0.2$，$\lambda = 4$ 和 $i_0 = 1/n$，看起来像是在图8-6中的一个平滑的逻辑斯蒂曲线。$i(t)$ 对 t 的相同的离散模拟绘图中，其中 $\gamma = \Delta = 0.2$，$\lambda = 12$ 和 $i_0 = 1/n$，看起来像一个混沌的锯齿曲线。这两种传染病的唯一区别是连通性λ。连通性越高，解中的不稳定性就越多。随机网络中不稳定性随着密度的增加而增加，因为λ是密度的函数：$\lambda = (n-1) \times$ 密度。

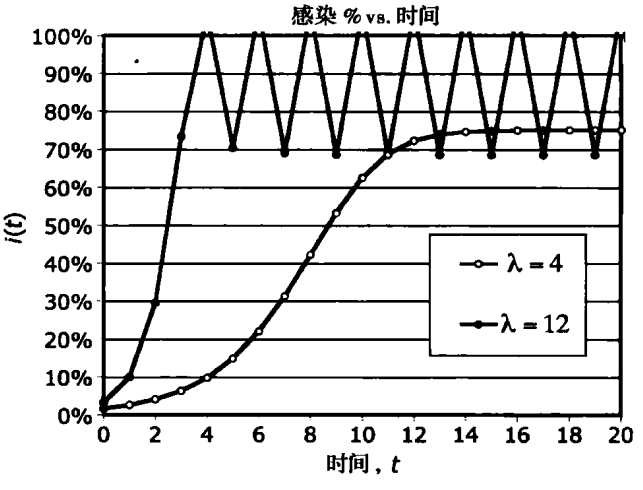


图 8-6 在任意的两个不同密度上 KW 模型与时间的离散模拟，$\lambda = 4$ 和 12。更高度网络导致了混沌解

因节点不同而带来的度值上的很大差异以及局部的感染率变化导致了波动。节点度在随机网络中是一致的，但在无标度网络中就不一致。因此，我们希望在具有高度节点的非均匀网络中观测到振荡。

⊖ 某些作者定义 $\tau = \gamma/\Delta$，因此持续传染病在阈值以上发生。

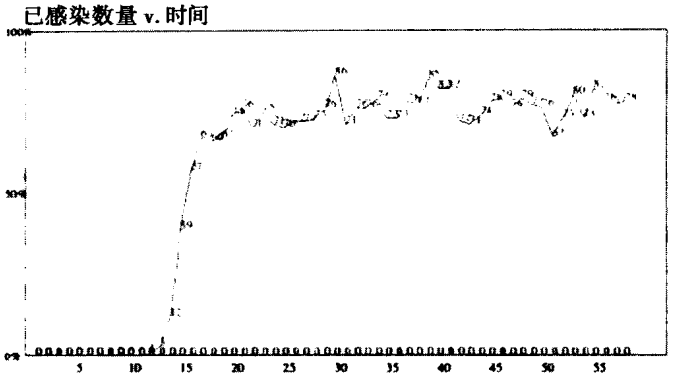


图 8-7 对于随机网络 $n = 100$，$\lambda = 4$，$\gamma = \Delta = 20\%$，$i_0 = 0.01$（$\tau = 25\%$），SIS 网络传染病与时间的关系图

图 8-7 演示了在 $\tau = 25\%$ 和连通性为 $\lambda = 4$ 的大约 $(1 - 25\%) = 75\%$ 确定性固定点解的振荡。已感染节点密度根据逻辑斯蒂增长曲线增长、下降，然后在固定点密度附近振荡。固定点的高度正如 KW 模型预测的一样。

$$\frac{1-\Delta}{v} = \frac{1-0.2}{(0.2)(4)} = 0.75$$

图 8-7 还演示了具有统一随机连通性的网络中的持续稳定的 SIS 传染病。当度序列分布不均匀时会发生什么呢？例如，对于具有平均度序列 $\lambda = 4$ 的无标度网络，传染病是否会到达同样的固定点？无标度网络的度序列分布是不均匀的，所以答案是否定的。为了更好地理解答案，我们需要一个集成更多网络拓扑信息的精确的模型。

Kephart 和 White 证明了除非已感染个体感染其他个体的速度超过了个体康复的速率，否则持续稳定的传染病就不会发生（Kephart，1991）。按照上面得出的解，这就是说持续稳定的传染病会在 $\tau = (\Delta/v) < 1$ 的随机网络中发生。这个结论忽略了导致传染病非零概率消失的波动。即使在 $\tau < 1$ 时，由于概率效应和混沌负效应也不能保证传染病保持稳定。但是 KW 认为当感染率能够远远超过阈值时，那么就越有可能保持稳定。特别是他们提出了在随机网络中 SIS 消亡的概率是 $\rho(0,t) = \tau^{I(0)}$，这里 $I(0)$ 为初始感染的个体数。典型情况下 $I(0) = 1$，所以随机网络中传染病持续稳定的概率是 $(1 - \tau)$。因此，稳定概率和固定点值随着感染率增长而增加，因为 $(1 - \tau)$ 会随着 γ 的增加而增加。感染率可以取阈值到 100% 人群值之间的任意值：$\tau \leqslant \gamma \leqslant 1.0$。

8.2.2 一般网络中的传染病阈值

只有极少数网络是随机的，因此 KW 模型在实际网络中的应用很有限。那是否能找到在任意网络中传染病传播问题的解呢？答案是肯定的，本节将给出。我们沿着 Z. Wang、Chakrabarti、C. Wang 和 Faloutsos 的推导（WCWF 模型）——他们首先提出任意网络中的传染病阈值的一般模型和固定点密度。WCWF 模型令人印象十分深刻，因为该模型使用网络谱半径，可以很好地找到任意网络中的传染病阈值，而忽视它的结构问题。WCWF 证明网络传染病阈值为 $\tau = \rho$，这里 ρ 为网络的谱半径，而 $\tau = \Delta/\gamma$ 如上述所定义[⊖]。这里给出了网络传染病的一般理论，并很好地回答了与自然界传染病和网络传染病之间差异有关的一些最终问题。

SIS 传染病一般是用谱半径来衡量的，这是一个比平均度 λ 更好的网络拓扑测量方式。WCWF 模型是对 Kermack-McKendrick 非线性方程的线性近似。线性化在大感染率的情况下通常

⊖ WCWF 定义 $\tau = \gamma/\Delta = 1/\rho$，但是我们使用倒数定义，以便与 Kephart 和 White 保持一致。

不够精确，然而对相对小的感染率却能得出很好的结果[⊖]。阈值的准确数值涉及非线性方程的求解。我们建议一个针对真实的非线性模型的二次方程，二次方程近似于稍稍改进了 WCWF 模型，特别是针对大感染率 γ。

在任意网络中找到一般传染病模型的策略是：

1. 用公式表达将节点 i 在时间 t 受感染概率 $p_i(t)$ 与它的 degree$(i) = k_i$ 关联的状态方程，此时邻近节点 $j = 1,2,\cdots,k_i$ 可能会、也可能不会感染节点 i。

2. 通过假设一些小的概率（将此称为 WCWF 近似）来线性化状态方程，然后通过将感染的非线性概率替换成二次模型来改善近似法，以便得到大概率下的阈值 τ 的表达——将这种称为改进的 WCWF 模型。

3. 通过总结单节点状态方程，得出表示所有节点感染概率的系统方程的 WCWF 近似。这将导出系统矩阵 $\boldsymbol{S}$，它通过系统方程与邻接矩阵 $\boldsymbol{A}$ 相关。系统矩阵 $\boldsymbol{S}$ 和邻近矩阵 $\boldsymbol{A}$ 的谱分解导出与感染率 γ 和 Δ 相关的阈值的标量方程。

4. 利用当传染病的最大系统方程特征值小于 1 时网络传染病消亡的事实，找到传染病阈值，$\tau = \Delta/\gamma = \rho$，这里 ρ 是邻接矩阵 $\boldsymbol{A}$ 的谱半径。修改这个关系以便获得非线性的 WCWF 方程，$\tau = \rho[B - A\gamma]$，A 和 B 为参数。

不像 Kermack-McKendrick 那样将感染密度 $i(t)$ 建模成时间的函数，WCWF 将节点 i 在时间 t 感染的概率 $p_i(t)$ 建模成直接被邻接节点在时间 $(t-1)$ 感染的结果。假设 $j = 1,2,\cdots,k_i$ 是节点 i 的具有度为 k_i 的邻接节点，而 γ 和 Δ 与之前一样是感染率和恢复率。如果节点 i 在时间 $(t-1)$ 没有被感染，那么在时间 t 的感染概率为被 k_i 邻接节点感染的概率的总和。

节点 i 在时间 $(t-1)$ 不被感染的概率为 $\gamma[1 - p_i(t-1)]$，被邻接节点感染的概率是 $\gamma\sum_j^{k_i} p_j(t-1)$，因此感染的联合概率为：

$$\text{被邻居感染的概率为：}\gamma[1 - p_i(t-1)]\sum_j^{k_i} p_j(t-1)$$

如果节点 i 在时间 $(t-1)$ 已经被感染，那么它在时间 t 仍旧被感染的概率就是它在时间 $(t-1)$ 被感染和不会康复的概率的联合概率，也就是 $(1-\Delta)$ 乘以 $p_i(t-1)$：

个体感染后而不能康复的概率为：$(1-\Delta)p_i(t-1)$

将两个事件求和就得到节点 i 在时间 t 感染的概率：

$$p_i(t) = \gamma[1 - p_i(t-1)]\sum_j^{k_i} p_j(t-1) + (1-\Delta)p_j(t-1); i = 1,2,\cdots,n$$

注意感染密度就是每一个时间步 $p_i(t)$ 的总和：

$$i(t) = \sum_i p_i(t)$$

相对于状态而言，我们对节点的概率行为更感兴趣，因此我们进行到了策略的第二步。通过说明邻接节点概率的总和刚好与列矢量 $\boldsymbol{P}(t)$ 乘以邻接矩阵 $\boldsymbol{A}$ 的行相同，状态方程中的第一项 $p_i(t)$ 被转换为矩阵形式。矢量 $\boldsymbol{P}(t)$ 是列矢量 $[p_1(t),p_2(t),\cdots,p_n(t)]^{\mathrm{T}}$。邻接矩阵 $\boldsymbol{A}$ 中 i 行中的值与节点 i 的邻居刚好相关。这个邻接矩阵是对称的，因此 $\boldsymbol{AP}(t)$ 刚好是以上等式的总和。

因此，状态方程的矩阵形式是：

$$\begin{aligned}\boldsymbol{P}(t) &= \gamma[1 - p_i(t-1)]\boldsymbol{AP}(t-1) + (1-\Delta)\boldsymbol{P}(t-1)\\ &= \{\gamma[1 - p_i(t-1)]\boldsymbol{A} + (1-\Delta)\boldsymbol{I}\}\boldsymbol{P}(t-1)\end{aligned}$$

这里 $\boldsymbol{I}$ 是 $n \times n$ 维单位矩阵，$\boldsymbol{A}$ 为网络的邻接矩阵。

⊖ 为了本章的目的，$\gamma > 10\%$ 被认为是“大的”感染率。

除了项 $[1-p_i(t-1)]$ 外（因为该项并不能很好地适用于矩阵公式中），上述状态方程是理想的。WCWF 假设小的概率，因此该项可以被 1 替代而不会损失太多精确性。这在很大程度上简化了解，因为我们现在就可以将系统矩阵 $\boldsymbol{S}=\{\gamma\boldsymbol{A}+(1-\Delta)\boldsymbol{I}\}$ 分解成为它的特征值。假设 WCWF 近似值是有效的，那么状态方程变成

$$\boldsymbol{P}(t)=\boldsymbol{S}\boldsymbol{P}(t-1)\text{；给定 }\boldsymbol{P}(0)$$

一阶差分方程的解为：$\boldsymbol{P}(t)=\boldsymbol{S}^t\boldsymbol{P}(0)$。

系统矩阵 $\boldsymbol{S}$ 的谱分解产生了由邻接矩阵 $\boldsymbol{A}$ 表示的网络连通性和传染病参数 γ 和 Δ 之间的关系。设 $\rho(S)$ 为 $\boldsymbol{S}$ 中的最大非平凡特征值（谱半径），并设 $\rho(\boldsymbol{A})$ 为 $\boldsymbol{A}$ 中的最大非平凡特征值（谱半径）。那么系统状态方程的对角线版本为：

$$\gamma\rho(\mathrm{A})+(1-\Delta)=\rho(\mathrm{S})$$

当 $\rho(\boldsymbol{S})<1$ 时，系统状态方程解 $\boldsymbol{P}(t)=\boldsymbol{S}^t\boldsymbol{P}(0)$ 会接近于 0，否则到达某一固定点。因此，对于 $\rho(\boldsymbol{S})=1$ 传染病阈值是可以实现的：

$$\gamma\rho(\boldsymbol{A})+(1-\Delta)=1$$

求解 Δ 然后被 γ 除就得到 τ：

$$\Delta=\gamma\rho(\boldsymbol{A})$$

$$\tau=\frac{\Delta}{\gamma}=\rho(\boldsymbol{A})$$

当 $\tau=\Delta/\gamma\leqslant\rho(\boldsymbol{A})$ 时传染病持续，否则消亡。通过 (ρ,Δ)，$\rho>0$，$\Delta>0$ 定义的半无限平面，在时间 τ 内所有落入该区域阈值线之上的传染病消亡了，而所有该阈值线以下的则持续存在下去。这个结果证实了 WCWF 模型，它定义了任何网络下的传染病阈值为[⊖]：

$$\tau=\frac{\Delta}{\gamma}=\rho(\boldsymbol{A})\text{ 或 }\Delta=\gamma\rho(\boldsymbol{A})$$

在 Δ 和 γ 之间的关系沿着阈值线分成持续传染病和非持续传染病。

网络中传染病阈值一般问题的良好解，对小的感染率来讲是精确的，但在具有高感染率的相对较小的网络里会发生什么呢？

思考以下 $[1-p_i(t-1)]<1$ 或 $p_i(t-1)\gg0$ 的非线性 WCFW 模型。从 Kermack-McKendrick 和 KW 模型中，我们期望感染概率服从 S 形逻辑斯蒂增长曲线，而这个曲线是由 $i(1-i)$ 或 $\gamma(1-\gamma)$ 非线性地推导出的。假设我们通过将二次方程 $\gamma(B-A\gamma)$ 代入到 WCWF 模型中来近似这个非线性，然后看看会得到什么。这就形成了如下修改的特征值方程：

$$\gamma(B-A\gamma)\rho(\boldsymbol{A})+(1-\Delta)=1$$

像前面一样求解 Δ 和 τ，我们得到：

$$\tau=\frac{\Delta}{\gamma}=\rho(\boldsymbol{A})(B-A\gamma)\text{ 或 }\Delta=\rho(\boldsymbol{A})\gamma(B-A\gamma)$$

通过拟合修改过的 Δ 表达式来模拟图 8-8 中所示数据，我们可以得到参数 A 和 B。表 8-2 总结了结果。图 8-8 画出了恢复率 Δ 和感染率 γ 之间的关系（沿着阈值线 $\tau=\rho(\boldsymbol{A})(B-A\gamma)$，针对很多不同大小和类别的网络）。在阈值线之上传染病消失，阈值线之下传染病持续保持。注意 τ 是感染率 γ 的函数。之后我们建议一种替换等式，其中 τ 是 Δ 的函数。

在图 8-8 中，对于较小的 γ 值，我们观察到 Δ 和 γ 之间的近似线性关系。这就证实了对于小概率感染的 WCWF 近似。但随着 γ 的增加而线性衰减——以由参数值 A 和 B 确定的二次速率消失。

⊖ WCWF 采用倒置版本：$\tau=\gamma/\Delta=1/(\rho(\boldsymbol{A}))$。

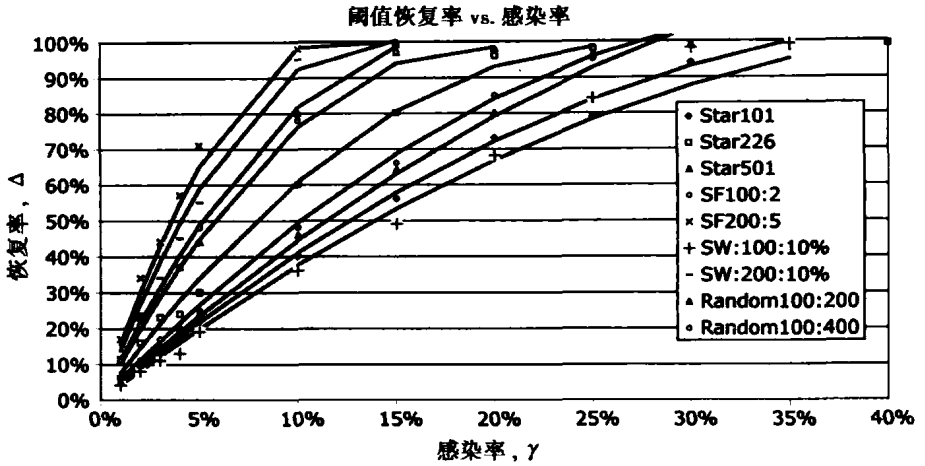


图 8-8 $A(\gamma,\Delta)$ 绘图：随机、无标度、小世界和星形网络恢复率 Δ 和感染率 γ 之间的关系。每条曲线之下为持续传染病区域，每条曲线之上代表传染病消失的 (γ,Δ) 区域

表 8-2 图 8-8 的曲线拟合参数

	ρ	λ	A	B
SW100:200	4.17	4.00	1.000	1.000
Star101:100	10.00	1.98	0.500	0.460
Rand100:200	4.94	4.00	1.000	1.000
SF100:2	7.37	3.94	1.000	0.770
Star226:225	15.00	1.99	0.950	0.500
Star501:500	22.36	1.99	1.200	0.460
Rand100:400	9.06	8.00	3.500	1.250
SW200:1000	10.13	10.00	5.000	1.410
SF200:5	15.85	9.85	4.000	1.020

二次近似对 WCWF 模型有所改进，但由于过于复杂而显得不尽如人意。阈值 τ 不再是谱半径的简单函数，而是依赖于 $\rho(A)$ 、γ 和 λ。事实上，精确的非线性函数关系一直是一个没有得到解决的问题。

我们可以将表 8-2 中所列出的仿真网络排序并显示在图 8-8 中，根据阈值线将图 8-8 中的二次曲线分成持续和非持续区域。从右（最少持续）至左（最多持续）查看图 8-8，网络按序排列，就像表 8-2 中的排列一样（第一个数为 n，第二个数为 m 的用于随机和小世界网络，而第二个数为 Δm 的用于无标度网络）。例如，最少持续传染病发生在 SW100：200（$n = 100$，$m = 200$ 的小世界网络），最多的则发生在 SF200：5（$n = 200$，$\Delta m = 5$ 的无标度网络）。

一般来讲，网络传染病随着它们的网络主机谱半径的增加而变得越持久，但是连通性同时也起着作用。这个事实可以在图 8-2 中明显看出，其中的网络排序由 ρ 和 λ 值决定。特别地，A 和 B 可能（很差地）近似成谱半径和连通性的函数——并不只是谱半径[⊖]。

对于图 8-8 中的网络仿真以及表 8-2 中的总结，我们观察到 A 和 B 随着连通性的增加而增加，而随着谱半径的增加而减少。例如，$A(\rho,\lambda)$ 直接和 λ 成正比，但跟 ρ 成反比，然而 $B(\rho,\lambda)$ 却仅与 ρ 成反比。下面的关系是由曲线拟合决定的。图 8-9 中画出了 A 和 B 的模型方程与表 8-2

⊖ 我们假定非线性的情况下，其中 γ 允许“很大”。否则，谱半径就是主导因素。

中给出的经验值的关系：

$$A(\rho,\lambda) = \lambda\left(\frac{2}{\rho}+\frac{1}{4}\right) \sim O\left(\lambda+\frac{\lambda}{\rho}\right)$$

$$B(\rho,\lambda) = \frac{1}{\rho}+\frac{1}{4} \sim O\left(\frac{1}{\rho}\right)$$

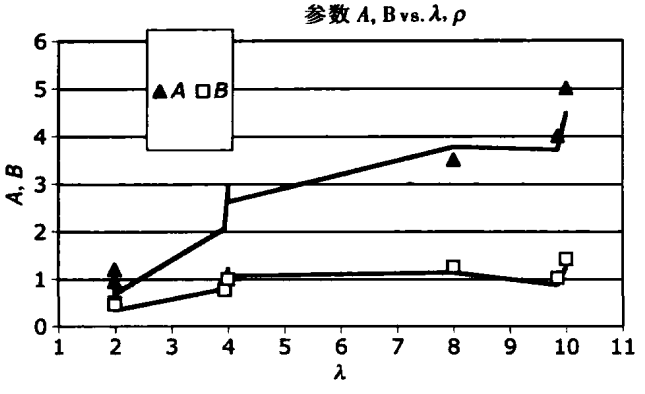


图 8-9 对于图 8-8 中的仿真网络，参数 $A(\rho,\lambda) = \lambda(2/\rho) + (1/4)$ 和 $B(\rho,\lambda) = (1/\rho) + \frac{1}{4}$ 与 λ 的关系

例如，对于 $n = 100$，$\Delta m = 2$ 和 $\gamma = 10\%$ 的无标度网络，阈值 τ 是多少呢？从表 8-2 中的表项 SF100：2，我们得到了谱半径 $\rho(A) = 7.37$，连通性为 $\lambda = 3.94$，$A = 1.0$ 和 $B = 0.77$。把参数带入到上面得到的等式中，就得到阈值恢复率 Δ 和阈值 τ：

$$\Delta = \rho(A)\gamma(B - A\gamma) = (7.37)(0.10)((0.77 - 1.0)(0.10)) = 49.4\%$$

$$\tau = \frac{\Delta}{\gamma} = \frac{49.4\%}{10\%} = 4.94$$

仿真为等价的无标度网络产生近似的 $\Delta = 48\%$ 和 $\tau = 4.8$，因此改进过的 WCWF 估计非常好。任何 (γ, Δ) 的组合导致 $\tau < 4.94$ 则产生持续的传染病；否则，传染病消亡[⊖]。

WCWF 近似好像对于“小的 γ”能非常好地适应，而改进后的 WCWF 近似则适用于“大的 γ”的传染病。尽管如此，这两种近似法对于极端问题却无法给出精确的解，例如非常小的网络、极度结构化或规则化的网络，或感染率极大的传染病。这些极端的例子通过每章末的几个例子加以探索。

改进后的 WCWF 公式近似用 Δ 来表示 γ，但一些应用可能需要刚好相反：用 γ 来表示 Δ。在这种情况下，求解 γ 的二次等式产生了可以替换的公式。该公式留给读者作为练习。

$$\gamma^2 - \left(\frac{B}{A}\right)\gamma + \frac{\Delta}{A\rho(A)} = 0$$

这个二次方程的非平凡根 γ_1 是 Δ 的函数，因此阈值以 Δ 表示就是：$\tau = \frac{\Delta}{\gamma_1}$。

传染病阈值由谱半径来决定，$t = \Delta/\gamma = O(\rho)$，并且我们知道星形网络的谱半径是 $\rho(\text{star}) \sim \sqrt{(\text{hub degree})} \sim \sqrt{(n-1)}$。因此星形网络的阈值感染率是：

$$\gamma(\text{star}) \sim \frac{\Delta}{O(\sqrt{(n-1)})}$$

随着 n 的无限增长，$\Delta/[O(\sqrt{(n-1)})]$ 会等于 0，因此 $\gamma_\infty(\text{star}) = 0$。SIS 传染病在无限星

⊖ 读者应该注意这些近似假定“中庸的”网络，不大可能采用极端的如完全网络、非常小的网络或非常大的网络。

形网络中肯定会持续存在下去！更现实的是，在阈值线附近，感染率与网络规模的平方根成反比。在无限的星形网络中，感染率大于0的传染病导致持续存在传染病，而与节点的恢复率Δ无关。

Pastor-Satorras 和 Vespignani（Pastor-Satorras，2001）首先提出了传染病在无限的无标度网络中肯定会持续存在下去。然后我们使用上述的星形网络加以演示，这种猜想对于任意无限网络似乎也是正确的！他们的断言是基于阈值参数随着连通性的无限增加而减少。在无标度网络中，$\rho(\text{scale-free}) \sim O(\sqrt{\langle \text{degree(hub)} \rangle})$，从前面对无标度网络的研究中，我们知道是 hub 相应于网络的规模成正比增加。因此，随着感染率变得任意小时，无限无标度网络也支持持续传染：

$$\gamma(\text{star}) \sim \frac{\Delta}{O(\sqrt{\langle \text{degree(hub)} \rangle})}，在极限下会接近于0$$

Pastor-Satorras-Vespignani 和星形网络研究结果显示出带有 hub 的网络中会有永不终止的传染病。事实上，这些 hub 被称做传染病学中的*超级传播器*，hub 的度越大，就越有动力持续下去；网络越大，星形或无标度网络的 hub 就越大。在本章的最后一节，我们说明了如何使用超级传播器以一种有效的方式来抵抗传染病。

8.2.3 一般网络中的固定点感染密度

图8-10描绘了稳定状态固定点感染密度作为图8-8中研究的网络的谱半径的函数。这些网络谱半径范围值从2到22，网络大小为100到501个节点。网络密度是稀疏的——以少数几个百分比点的数量级。最佳拟合直线最好地粗略拟合了这些仿真中得到的数据。

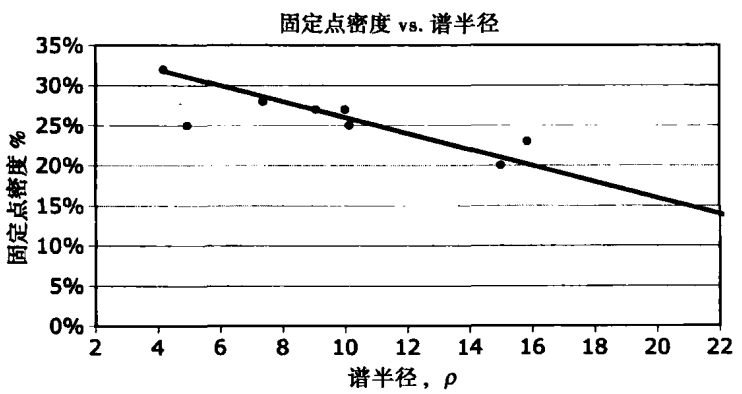


图8-10 图8-8中仿真网络的固定点密度的线性近似值与谱半径ρ

为什么随着谱半径的增加固定点感染却在下降？直觉告诉我们刚好与此相反，但不要忘记这些数据是在传染病*接近阈值*处得到的。随着ρ（network）的增加，Δ也在增加，并且随着Δ的增加，恢复个体与感染个体的比率按照$\Delta/\gamma = \rho$（network）增长。Δ的增长推动了感染个体的下降，因此随着谱半径的增加固定点感染则下降。

接近阈值τ的线性关系是$O(-\rho)$：

$$i_\infty = b - m\rho, \text{当最小二乘参数 } b = 0.36, m = 0.01 \text{ 时 } = 36\% - \left(\frac{\rho}{100}\right)\%$$

一般来讲，ρ随着连通性λ（也是直接与网络的密度成正比）的增加而增加，因此，连通性越大，感染节点的固定点密度就越低。换句话来讲，密集网络要比稀疏网络能更好地维持低等级的感染存在。这对于人类和动物群体有着重要的意义——密集更容易维持低等级的持续感染。

或许图8-10中的最有趣的属性是所有网络类都满足$i_\infty(\rho)$关系。谱半径看起来是SIS网络传染病研究中使用的优秀度量标准，但谱半径不是确定阈值的唯一因素。像我们说的那样，阈值τ

随着传染病速率增加到超过了两位数的百分比而变得非线性。

8.3 网络传染病仿真软件

前面的分析主要依靠仿真获得参数值并研究各种网络上的传染病行为。这些仿真是通过程序 Network. jar 来执行的，通过实现图 8-2 的状态机实现网络内的传染传播。

方法 NW_doEpidemic(boolean SIR)如果 boolean SIR 为 true 则实现 SIR 状态机，如果参数 SIR 为 false 则实现 SIS 状态机。算法有两部分：（1）扫过所有链路从链路的一端向另外一端传播感染；（2）扫过所有节点以便确定它们是否被恢复或者被删除。

易感节点颜色为白色，感染节点颜色为红色。恢复的节点在 SIS 中为白色，在 SIR 中为黄色，删除的节点为黑色（SIR）。某些额外的兴趣点为：

1. 方法 NW_doEpidemic( )通过重新洗牌网络的链路随机化过程顺序，因此不存在由于链路次序的仿真偏差。随机化是由类 Shuffler（仿真牌的重洗）来执行的。

2. 基本的传染病算法很简单，为每条链路检查头部和尾部节点并检查这两者是否被感染。如果一端被感染了而另外一端没有，就以概率 γ = InfectionRate 感染易感节点，颜色变为红色。使用输入值 t_r = RecoveryTime 设置新感染的节点的计时器。该计时器每个时间步就会减 1 直到零为止。感染（红）节点既可以以概率 DeathRate 消亡或者当它们的计时器到零时恢复。当 SIR 为 true 时，恢复节点颜色变为黄色。

3. 对于 SIS 传染病，当 RecoveryTime 大于零时，算法防止刚刚感染的节点在感染时的同一时间间隔恢复。这会对结果造成影响。想要允许感染节点立即以概率 RecoveryRate 恢复的用户将在 Network. jar 的偏好设置面板上将 RecoveryTime 设置为零。

4. 对于 SIR 传染病，传染节点不会消亡（被删除）或恢复（被恢复）直到计时器到零为止。当计时器达到零时，感染的节点以概率 DeathRate 被删除，或者以概率 RecoveryRate 被删除。

该代码是通过按 Network. jar 控制面板上的 INFECT 按钮来激活的。感染密度与时间关系图也会显示出来，显示各种参数，例如当前、平均和峰值感染密度。

```
public void NW_doEpidemic(boolean SIR){
     //Randomly Propagate the infection
     Shuffler random_list = new Shuffler(nLinks);
     for(int i = 0; i < nLinks; i++) random_list.card[ i] = i;
     random_list.SH_Shuffle(nLinks);              //Scramble link order
     //Part 1: Propagate infection via links
     for(int i = 0; i < nLinks; i++){
          Node n = null;              //Target of infection
          int j = random_list.card[ i];
          int tail = Link[ j] .tail;
          int head = Link[ j] .head;
          if(node[ tail] .color == Color.red        //From tail to head
                    &&
             node[head].color == Color.white) n = node[head];
          else if(node[ head] .color == Color.red  //From head to tail
                    &&
             node[tail].color == Color.white) n = node[tail];
           if(n != null && 100*Math.random() <= InfectionRate) {
                n.color = Color.red;              //Infect!
                n.timer = RecoveryTime;           //SIR
           }
     }
```

```
        //Part 2: Handle SIR vs SIS Change in Red(infected) nodes
        for(int i = 0; i < nNodes; i++){
             Node n = node[i];
             if(n.color == Color.red){
                if(SIR) {                                  //SIR model
                   n.timer-;
                   if(n.timer <= 0){
                      if(100*Math.random() <= DeathRate) {
                          n.color = Color.black;       //SIR removed
                              }
                              else {
                                 n.color = Color.yellow; //SIR recovered
                              }
                          }
                   }                                       //SIS model
                   else if(100*Math.random() <= RecoveryRate) {
                      if(n.timer == 0)
                      n.color = Color.white;          //SIS recovered
                      else n.timer = 0;               //1-shot
                    }
             }
          }
    }//NW_doEpidemic
```

8.4 对策

之前的分析解释了传染病是如何传播的，但却不能解释传染病是如何停止的。总的来讲，我们想知道如何尽可能快地根除传染病并减少感染个体的数量。

这个重要的问题在很多领域都有应用：

1. 在生物有机体（公共卫生）中，我们如何限制疾病的传播呢？
2. 我们如何从互联网（计算机科学）上去掉不需要的病毒呢？
3. 我们如何阻止基础设施系统中故障的传播（电网和电信等的层叠式故障等）呢？
4. 我们如何对付竞争对手的理念、产品、时尚的传播呢？
5. 我们如何消除那些讨厌的理念、概念、新闻或政治思想（的传播）呢？

这些问题根据策略的不同有着不同的答案。我们选择研究*病毒式营销*技术的应用作为一种阻滞网络传染病传播的一个有趣的方式。这个想法很简单，释放一个对策传染源（称之为*抗原*），然后让其像传染病一样传播到网络的所有节点上。每当抗原遇到一个感染的节点时，它中和掉感染，接着继续传播。抗原行为表现就像是传染病一样具有同样的速率并恶意地进行传播。尽管如此，仍然有一个重大的区别——抗原不是用来删除的。它不会消失掉，但是它可能会达到一个固定点密度，就像其他传染病一样[⊖]。

对策抗原在某一点开始（被称为*接种节点*），之后以与传染病相同的速率 γ 传播。仿真通过程序 Network. jar 实现，感染节点被涂成红色，抗原节点为蓝色，中性节点为黄色。与之前的一样，易感节点是白色，但是值得一提的是，抗原通过白、红、黄色节点传播。

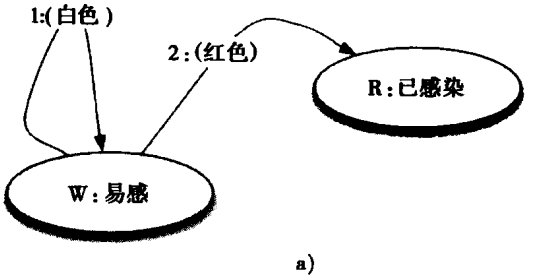
我们提出的问题是："发起对策的最佳位置在哪里？"我们将"最佳"定义成"时间最少"和"最低的感染峰值"，并且推测最好的策略依靠于抗原是从哪里开始注入。在本节中，我们对

⊖ 我们将抗原定义成能够消除免疫反应的传染，例如中性化感染过的节点。

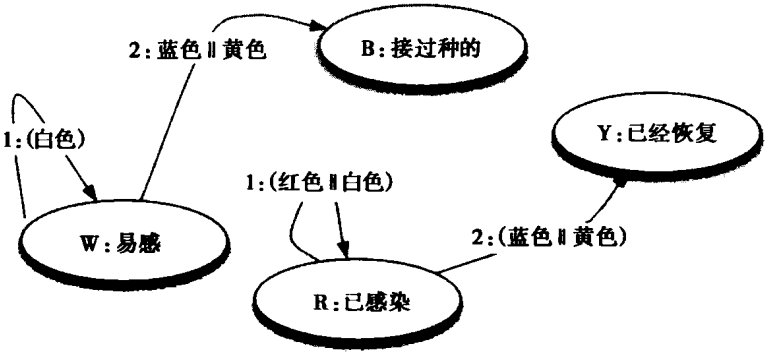
比应用于任意网络的4种策略或对策：（1）随机注入接种；（2）hub注入接种；（3）中心节点注入接种；（4）最大紧度节点注入接种。进一步来说，我们探索以上每个策略的行为和在随机、无标度和小世界网络中阻滞传染病的有效性。

8.4.1 对策的算法

图8-11显示了一个状态图，用于传染病之间的红蓝抗原竞争，传染病用红色表示，抗原由蓝色表示。在图8-11a中，红色传染病以概率γ将白色易感个体转换成红色感染个体。红色个体将保持为红色直到蓝色抗原对它“治疗”为止，即将它转换成为黄色。



a)



b)

图8-11 红－蓝抗原仿真状态图：a）红色传染病传播；b）蓝色抗原的扩散。迁移标记成邻接节点的颜色。所有的迁移以概率γ发生

没有抗原对策的话，所有节点最终将被感染，因为在这个模型里没有恢复处理。相反，如果蓝色抗原在个体被感染为红色个体之前变成了白色易感个体，白色个体节点将以概率γ转换成接种过的蓝色节点，并永久保持为蓝色。

图8-11b中的状态图要比图8-11a更为复杂，因为白色节点只有在邻近蓝色或黄色节点时才以概率γ迁移转换成蓝色节点。类似地，只有当红色节点邻近蓝色或者黄色节点时，红色节点以概率γ迁移转换成黄色节点。一旦节点到达蓝色或黄色节点，将会保持在那个状态。我们假设红色和蓝色传播源都以相同的速率传播，我们会问：“注入最初的蓝色抗原以便对红色传染以最有效的方式加以限制的最好节点是哪个?”

在下面的仿真中，两种传染病扩散过程在感染扩散（图8-11a）和抗原扩散（图8-11b）之间交替进行。一个随机选择的最初感染的节点，通过它的接触传染传播到邻近的邻居，随后是从注入节点开始的对策抗原。Java代码通过随机选择网络链路避免任何处理上的偏差。经过每一步骤后，它检查了每一个邻接节点并借助于状态图以便决定它的下一个状态。

8.4.2 接种策略对策

运用抗原去阻止病毒的想法多少有些新潮，这和在两个物种（一个红色感染物种和一个

蓝色抗原物种）之间建立竞争有些相似，让更强大的物种取胜。如果成功，这将提供一个战胜互联网病毒以及阻止致命的人类病毒的全新方法。这个方法是以毒攻毒，但最好从哪里开始呢？

我们沿着与 Chen & Carley 相似的策略（Chen, 2004）。但是，我们的策略在于学习四种简化了的接种算法：（1）随机——随机选择一个接种节点；（2）hub——接种最大连接数的节点；（3）中心节点——为我们找到的第一个离所有其他节点最短距离的节点接种；（4）紧密节点——为大多数路径通过的中介节点接种。

我们猜想 hub 接种算法给予抗原一定优势，因为它要比其他任何节点具有更多邻接的节点，并且中心节点接种算法离所有其他节点具有更少的跳数，从而因此应该在传染病到达之前到达所有其他节点。结果证明，hub 接种要比其他三种更好，但是令人惊讶的是，最大紧度算法接近等于 hub 接种算法。

一个最有效的接种算法的标准是，要么在最短时间内清除红色感染表示的传染病，要么最小化峰值感染密度。接种算法尽可能快地删除传染病以便满足时间目标，最小峰值感染目标是通过在其峰值感染最少数目的节点算法来得到满足。

很多运用 Network. jar 程序的仿真结果都在图 8-12 中列出显示。图 8-12 中，在随机、无标度（$\Delta m = 4$）和小世界网络（5% 重联概率）对于具有 $n = 100$ 个节点和 $m = 400$ 条链路、感染率为 10% 的网络中的删除时间进行了对比。令人惊讶的是，最大紧度算法也一样，在某些情况下要比 hub 接种算法甚至还要略好一些。为什么？正如结果证明，紧密节点也是除了小世界之外所有类网络的 hub。

从图 8-12 中，我们可以得出如下结论：

1. 最佳（最低）的总删除时间是通过 hub 接种算法来实现的，这证实了直观感觉是正确的。尽管如此，hub 接种算法对无标度网络来讲没有紧度接种算法效果好。但是，这个差异不是很大。

2. 删除之前的最佳（最低）的总峰值感染是通过 hub 接种算法来实现的，特别是在无标度网络中最有效，但在小世界网络中不那么有效。对于无标度和小世界网络，紧度接种算法比 hub 接种算法效果要稍好一些，但差异也不是很明显。

3. 当网络是随机时，四种算法的消亡速度差异不大，但是峰值感染百分比率却刚好相反。随机和中心接种是在所有情况下表现最差的算法。

正如前面所提到那样，这些结论在大多数学科中都很重要，特别对互联网的保护有着深远的意义。目前流行的是在局部层次通过在台式和笔记本计算机上安装杀病毒软件，但这远不如强化互联网的 hub 更有效。hub 处理了全部流量的大百分比，因此强化几十个 hub 服务器远比强化数百万计客户机更有意义。令人惊讶的是，紧度接种算法也证明是一种有效的对策。在互联网上，最大紧度意味着与其他的节点相比会有更多路径通过紧密节点。例如，最大紧度路由器或交换机比其他的处理更多的网络流量。因此，从这样的节点上发起对策更有意义。

这些结果导致得出这样一个结论，即清除最新的互联网病毒的最有效对策不是在个人客户计算机上，而是从几十个或最大的 hub 或最近的路由器或交换机上发布抗原。这个策略要比保护数百万计的桌面和笔记本计算机更加有效得多。尽管这个根本的结论与现行的计算机安全准则是相悖的，但确实有着一定意义，因为客户计算机既不是中心、最紧密的，也不是网络的 hub。

为网络安全采用抗原对策远比对付大量互联网病毒更加有效，但并不是所有情况都是如此。例如，并不总是很容易探测到恶意程序或者附件。一个能删除重要附件的抗原和恶意病毒一样有害。尽管如此，启用安全、简单的抗原是实惠、简单和明显有效的。

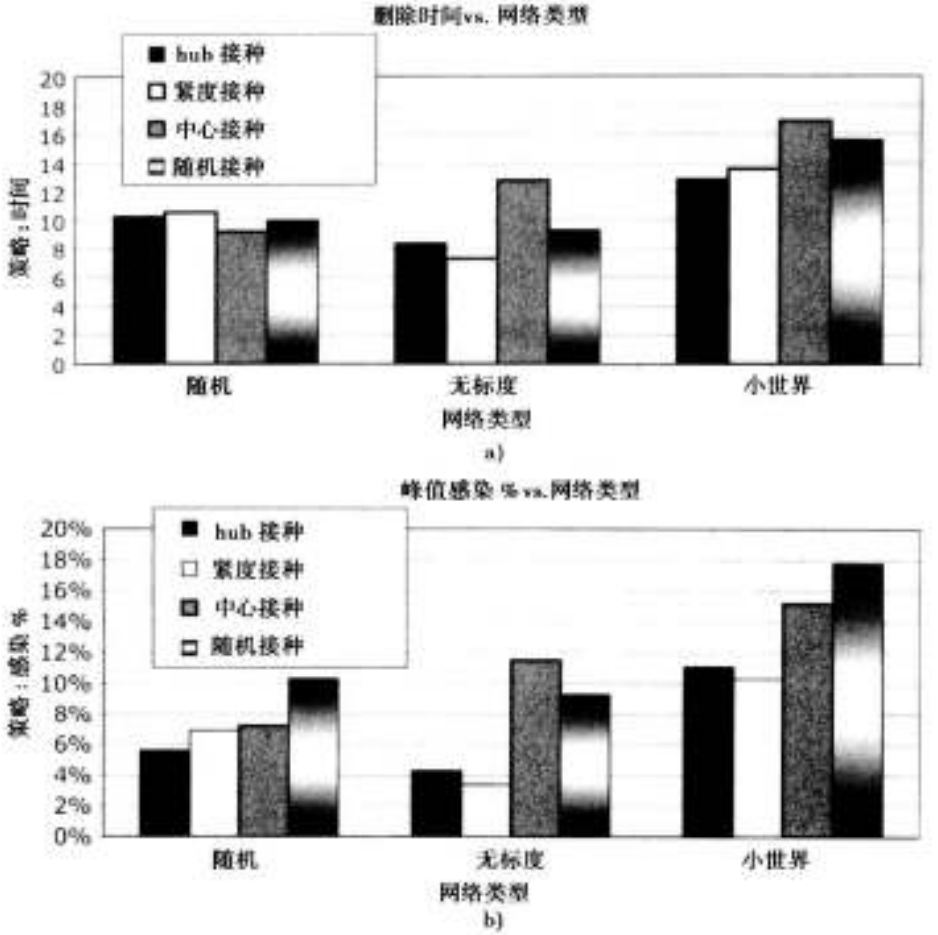


图 8-12 红－蓝抗原对策的结果仿真：a）最小时间目标（根除感染的时间）；b）最小峰值感染目标（对于随机、无标度和小世界网络）。网络属性：$n = 100$，$m = 400$，$\Delta m = 4$，重联概率 $p = 5\%$，感染率为 10%。每个数据点是通过平均 10 趟获得的

本章结论显示，hub 和紧度接种策略是对付任意大型网络中的传染病最有效的，无论它们是随机的、小世界的或无标度的网络。这个结论支持 Dezso 和 Barabasi 提出（Dezso，2002）的相似结论，他们提出了无标度网络中的随机分布式治疗策略是低效的。在最小连通因此也是最不关键的节点上的随机治疗是浪费的。与此相反，他们提出治疗 hub——验证了这里给出的仿真结果。我们已经证明，在最大紧度节点上接种也是有效的，在某些情况下甚至比 hub 接种还会更好，而与网络拓扑无关。

8.4.3 Java 抗原仿真

在本节中将阐述实现红－蓝抗原对策仿真的 Java 代码以获得图 8-12 的结果。参考图 8-11 中的状态图。在从 SETTINGS 菜单中选择接种策略后，通过按 ANTIGEN 按钮从程序 Network. jar 调用方法 NW_doAntigen( )。

在每个时间步 NW_doAntigen(boolean RED) 被调用两次：当 RED 为 true 时一次，紧接着 RED 又为 false 时又有一次。第一个实例根据图 8-11a 的状态图传播感染（红）；第二个实例根据图 8-11b 的状态图传播抗原（蓝）。这些代码都是直截了当的，下面情况除外。

随机顺序处理的感染和抗原都是通过链路传播的。链路的随机性是由方法 SH_Shuffle( ) 处理的。这个方法仿真了洗牌，因此方法 random_list. card[i] 返回了第 i 个随机的链路索引。

此时读者应该注意到了抗原是通过“下一个状态”变量 int_color——每个节点元素的组件来传播的。通过强制轮流阻止了红、蓝抗原之间不公平的竞争。例如，使用“下一个状态”变量防止抗原红在一个时间步从 A 传播到 B 到 C … 。相反，红色抗原可能会从节点 A 传播到 B，但之后它必须等到下一轮后才能传播到 C，依次类推。这种红与蓝传播交错是在两个阶段实施的——阶段 1 建立了“下一个状态”，而阶段 2 将下一个状态推进到下一时间步。以这种方式的交错避免在真实世界不存在的模拟偏差。

```
public void NW_doAntigen(boolean RED){
        //Randomly Propagate the infection
        Shuffler random_list = new Shuffler(nLinks);
        for(int i = 0; i < nLinks; i++) random_list.card[ i] = i;
        random_list.SH_Shuffle(nLinks);    //Scramble link order
        //Pass 1: calc next state
        int j;                             //Random link
        int tail, head;                    //End nodes
        if(RED){                           //Propagate red infection
          for(int i = 0; i < nLinks; i++){
          Node n = null;                  //Target of infection
          j = random_list.card[i];
          tail = Link[j].tail;
          head = Link[j].head;
          if(node[ tail] .color == Color.red //From tail to head
              &&
              node[head].color == Color.white) n = node[head];
          else if(node[ head] .color == Color.red  //From head to tail
                  &&
                  node[tail].color == Color.white) n = node[tail];
            if(n != null && 100*Math.random() <= InfectionRate) {
              n.int_color = 2;          //Infect! red
            }
          }
        }
          else for(int i = 0; i < nLinks; i++){  //Propagate blue
                                                 countermeasure
            Node n = null;                    //Target of infection
            j = random_list.card[i];
            tail = Link[j].tail;
            head = Link[j].head;
            boolean tail_color = node[ tail] .color == Color.blue
                                 ||
                    node[tail].color == Color.yellow;
            boolean head_color = node[ head] .color == Color.blue
                                 ||
                    node[head].color == Color.yellow;
            //Spread blue
            if(tail_color                  //From tail to head
                  &&
```

```
                node[head].color == Color.white) n = node[head];
            else if(head_color          //From head to tail
                      &&
                   node[tail].color == Color.white) n = node[tail];
            if(n != null && 100*Math.random() <= InfectionRate) {
                  n.int_color = 4;      //Inoculate! blue
             }
            //Recover yellow
            if(tail_color               //From red to yellow
                   &&
               node[head].color == Color.red) n = node[head];
            else if(head_color          //From head to tail
                      &&
               node[tail].color == Color.red) n = node[tail];
             if(n != null && 100*Math.random() <= InfectionRate) {
                  n.int_color = 3;      //Recover! yellow
             }
        }
        //Pass 2
        for(int i = 0; i < nNodes; i++) {
            switch (node[ i].int_color) {
                 case 1: { node[ i].color = Color.white; break;}
                 case 2: { node[ i].color = Color.red; break;}
                 case 3: { node[ i].color = Color.yellow; break;}
                 case 4: { node[ i].color = Color.blue; break;}
            }
        }
    }//NW_doAntigen
```

这个代码实现在图8-11中的状态图，简单来讲，从易感个体（白色节点）既可迁移到感染（红色）也可以迁移到接种（蓝色）个体，感染个体（红色）迁移到恢复个体（黄色）。一旦一个个体接种了，它将永久保持接种（蓝）状态。类似地，一旦一个个体恢复（黄）时，它也将保持恢复。因此，所有节点从易感（白）个体开始而终止于预防接种（蓝或黄）。

注意这种方法通过蓝和黄节点来传播抗原，也就是说，无论红节点何时出现在邻接的蓝或者黄节点附近，红节点将会以 γ 概率转换成黄节点。反过来的情况不会出现，即一个蓝节点绝不会转换到红或黄节点。事实上，蓝节点只有当白节点转换到蓝节点这种情况下才会发生。黄节点仅当在红节点转换为黄色节点（通过蓝或黄节点）时才会出现。

练习

8.1 假设 Kermack-McKendrick 模型和 $\Delta = 0$，$n = 100$，$i_0 = 0.01(1\%)$，$\gamma = 0.005$，那么需要多少时间才能感染全部人口的50%？

8.2 导出 Kermack-McKendrick 方程的随时间变化的解，当 $\gamma > 0$，$\Delta > 0$ 时。分别考虑两种情况：$\gamma > \Delta$，$\gamma < \Delta$：

(a) $\frac{\delta i}{\delta t} = \gamma si - \Delta i$

(b) $i(0) = i_0$

(c) $s + i = 1$

8.3 对于 $n = 100$，$m = 200$ 的随机网络，感染持续时间为10个单位，感染率=恢复率=20%，

近似的峰值感染密度为多少?

8.4 在具有表8-1给出的参数值的小世界网络,为了取得峰值感染密度不超过50%,连通性 λ 等于多少?

8.5 估计 $n = 200$, $m = 2000$ 时的小世界网络的谱半径。同等的无标度网络的谱半径等于多少?

8.6 已知 $n = 1000$, $m = 5000$ 的随机网络,传染病系数 $\gamma = 33\%$, $\Delta = 33\%$,近似固定点(传染平衡)密度是多少?

8.7 考虑具有 $n = 4$ 个节点并且 hub 在节点 0 的星形网络。进一步来讲,列矩阵 $\boldsymbol{P}(t-1) = [p_0, p_1, p_2, p_3]^{\mathrm{T}}$ 表示星形网络在时间 $(t-1)$ 被感染的概率,A 为星形网络的 4×4 邻接矩阵。说明 hub 节点 0 在时间 t 被它的邻居感染的概率 $p_0(t)$ 为矩阵相乘之积,$k[\boldsymbol{A} \times \boldsymbol{P}(t-1)]$,这里标量 $k = (1-p_0)\gamma$。也给定 $\gamma = \frac{1}{3}$,给出 $p_i(t-1) = \frac{1}{2}$,对于 $i = 0,1,2,3$,那么 $p_0(t)$ 的数值为多少?

8.8 对于 $\gamma = 5\%$ 持续感染大小为 $n = 65$ 的星形网络时,传染病最小恢复率 Δ 为多少?

8.9 对于以 $\Delta = 15\%$ 持续感染大小为 $n = 65$ 的星形网络时,传染病感染率 γ 的阈值为多少?

8.10 考虑完全连通的完全网络,$n = 100$,传染病感染率 $\gamma = 1\%$。那么阈值恢复率 Δ 为多少?比较 WCWF 近似与二次近似的仿真结果。

8.11 在练习8.10中,对于 $\rho = 99$ 和 $\gamma = 1\%$ 的完全网络,近似的固定点感染为多少?

8.12 当以下每种网络的大小无限增加时,阈感染率 $\gamma_0 = \frac{\Delta}{\rho(\text{network})}$ 会发生什么变化?

(a) 超环形网络
(b) 超立方网络
(c) 二叉树网络
(d) 随机网络

8.13 对于以下每种网络,哪种接种策略对减少删除清除时间最有效?

(a) 小世界
(b) 随机
(c) 无标度

8.14 对于以下每种网络,哪种接种策略对于减少峰值感染密度是最有效的?

(a) 小世界
(b) 随机
(c) 无标度

同　步

到此为止，我们已经从拓扑结构方面和节点的状态（感染与否）的角度学习了网络。本章扩展了节点状态的动态性质，并思考“什么性质的网络可以保证节点同步”。当所有的节点改变或者同时到达某一确定状态时，网络就同步了。否则，该网络就是不稳定的或混沌的。

1. 如果一个网络的所有节点值在这些节点的变化率趋近于零时收敛于一个常量，那么该网络就是同步的。

2. 一个动态网络如果其节点值是同步的，就被认为是稳定的；如果其节点值是振荡的，就被认为是瞬态的；如果其节点值在固定值和不稳定或混沌之间振荡等，则被认为是双稳态的。

3. 我们在网络中引入混沌映射以便查看节点的状态：如果一个网络最终同步，混沌映射中的每一个节点将会在混沌映射中找到一个确定点，称之为奇异吸引子——并且不会离去。混沌映射就是绘制 $t+1$ 时刻与 t 时刻的节点状态对比。

4. 我们证明如何通过固定一个节点来同步大部分任意网络（至少保持一个节点值为常数），附加一个三角循环（一个完整的三节点的子网），或通过控制链路形成至少一个三角循环来扩大混沌网络。

5. 通过一个生物同步的简单模型——鸣叫的蟋蟀，阐明拓扑结构对网络稳定性的影响。所仿真的蟋蟀社会网络中的个体，在聆听和鸣叫之间交替变化。如果在网络上存在一个循环，或其长度是一个奇数或至少固定住一个个体，那么鸣叫的蟋蟀是同步的（发声和听觉上是一致的）。额外添加一个三角循环能足以保证同步。一只占据主导的蟋蟀不自觉地在鸣叫和聆听之间变化就足够了。否则鸣叫的蟋蟀网络就不能同步。

6. 一个更一般的算法（Atay，2006）为每个节点分配了一个数值，并通过平均节点与其相邻的状态来计算状态变化。类似于蟋蟀社会网络，一个 Atay 网络至少包含有一个奇数长度的循环就会同步。如果将一个三角循环连接到一个任意节点上或是至少固定一个节点，则它就会同步。

7. 与当前小世界网络和生物系统中的同步如心脏起搏器理论相反，小世界网络不比其他类型的网络在同步上占优势。但是因为小世界网络自然地以簇的形式包含三角循环，所以它们被称为“自然同步器”。

8. 我们将基尔霍夫（Kirchhoff）网络定义成完全连通的网络，节点状态是其总流出量和总流入量之差。基尔霍夫网络中如果任何一对有向循环长度是互质数，则它就是同步的。

9. 我们将同步基尔霍夫网络理论应用到 Pointville 电力网络中，并且证实不管是将受损的链路还是将受损节点删除，只要留下的网络仍然包含有互质的循环长度，那么电网就是稳定的。否则，受损的网络就会混沌地响应而不能同步。这个性质是对实际的相当吸引人的近似——实际上电网会因为单条电路故障而变得不稳定，造成整个地区的停电。

简单的同步问题比比皆是，更复杂的网络同步现象是如互联网、电网、各种各样的反馈控制系统的人工动态系统。Network. jar 程序仿真以下的每种网络，并帮助深入研究其行为。

9.1 同步或不同步

许多网络科学应用涉及与节点或链路相关的状态或值，也包括网络拓扑结构。节点状态可以反映电网上的电压，神经系统上的电荷，或者停车场、水库、计算机内存容量。这些应用被建模成包含节点的动态网络，这些节点带有被称为状态的动态值。

关键问题是要问："动态网络的行为本质是什么?" 节点值是增加或者减少到无穷，还是直到它们逐步减弱到零为止？或者说，节点值是在固定值之间来回振荡的，还是到达一个确定值，然后无限地保持在那儿？这些都是稳定性的问题，网络同步性的研究就是网络稳定性的研究。我们要求一个稳定的网络在所有节点值同步时最终会达到稳定。也就是说，当所有的节点要么锁定步调一同前进，要么安静下来等于同一个值，网络就达到了稳定。

以锁定步调（lock-step）方式一同前进的网络被称为振荡，网络包含能到达同一个稳定值的节点就被称为同步。第三种可能存在的情况是，网络太不稳定了以至于网络中一部分节点的状态失控而不稳定，或者快速无限地增大、减小。这种不规律的网络是不稳定的。与最终的状态无关，这些节点会在稳定前很短的时间内表现得没有规律。我们将此行为称为混沌，因为状态改变不规律而且不可预测。

更正式地，动态网络 $G(t) = \{N(t), L(t), f(t)\}$ 是一个带有如下定义的附注节点和链路的图：

1. $N(t)$ 中的每个元素 v_i，用一个状态 $s_i(t)$ 来表示，下一个状态用 $s_i(t+1)$ 来表示，它们可用数值或任意性质（如颜色）定义。

2. $L(t)$ 中的每个元素 e_i，也可以表示为状态或下一个状态值，如流量或链路的容量。

3. 在这里研究的例子中映射函数 $f(t)$ 可以是动态的，也可以不是动态的，我们假设 $f()$ 是静态的（例如，网络的拓扑结构不随时间而变化）。

具有状态的动态网络是抽象的实体，它们可以代表物理设备，例如通信网络、电网或者个体和关系组成的社会网络的抽象。例如，我们演示蟋蟀的社会网络，通过模拟动物作为一个个体（节点）而将聆听邻接蟋蟀的鸣叫的动作作为链路。同样，我们模拟 Pointville 的社会网络，Flatland 被认为是带有对产品正、反意见的节点集合。这类网络对于那些想要通过口碑进行市场营销的促销来说很有用。

9.1.1 混沌映射

一个动态网络如果从一个初始状态 $G(0)$ 在有限的时间内演变成另外一个状态 $G(t^*)$，并且保持此状态，就被称为同步的。我们将 $G(t^*)$ 称为奇异吸引子，并且如果网络一直无限期地处于此状态，则称它为固定点。网络在没有明显的模式的情况下，反复从一个状态跳到另一状态，就被认为是混沌的。在两个或两个以上的奇异吸引子之间振荡的网络称为振荡器。振荡器在两个或多个状态下交替变化，也被认为是伪稳定。这样的网络同时具有混沌和稳定的行为。伪稳定节点通常如同一个双稳态振荡器一样振荡，或者像计算机中的触发器电路，所以我们也称它们为双稳态网络。

对于达到一个固定点的 G，G 中每个节点（和链路）的状态必须做同样的工作，从一个初始状态 $s_i(0)$ 开始，每个节点 $i = 1, 2, \cdots, n-1$ 必须达到最终状态 $s_i(t^*)$，并保持在这个状态，以使网络同步。一个双稳态节点在两个状态之间交替，并且一个不稳定的节点"失效"，它的状态变为无限的，或采取似乎是随机的值。我们称这些不稳定状态为混沌，尽管它们经常反复地重复相同的状态序列。

状态从 $s_i(0)$ 迁移到 $s_i(1)$、$s_i(2)$、$\cdots$、$s_i(t^*)$ 时在状态空间中形成轨迹（状态空间定义为

纵轴表示 $s_i(t+1)$、横轴表示 $s_i(t)$ 的混沌映射)。混沌映射仅是状态空间里的一个节点轨迹图。图 9-1 描述了稳定节点、双稳态节点和混沌节点的混沌映射。一个稳定节点被固定点所吸引，而不考虑从哪里开始。

双稳态节点在两点之间振荡，而混沌节点的轨迹是随机走动，或者就是明显的随机走动。一般地，一个混沌节点的轨迹描绘同样的不规则路径。这不是真正随机的，而是非线性的。

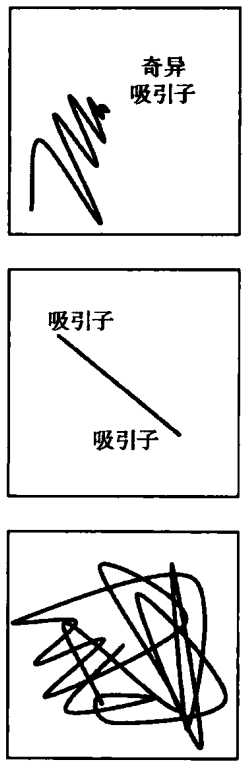


图 9-1 混沌映射——下列情况下的轨迹：a）到达奇异吸引子的稳定节点；b）在两个吸引子间振荡的双稳态节点；c）在状态空间内游荡的混沌节点

9.1.2 网络稳定性

尽管用术语“混沌”一词来描述节点或网络也许太牵强了，但是节点可能并非真的是混沌的，仅用它描述失控网络的古怪行为。这里我们是指节点值中的脉冲突起形式的干扰或损坏的链路形式的中断，导致邻接节点突然变化。节点不可能返回到它的奇异吸引子值。问题不在于中断是否改变节点或是网络的状态，而是该节点或网络能否恢复。稳定的网络就会恢复——或许非常慢，但不稳定的网络就不会。稳定性是我们用于描述节点或网络从它们的中断状态中恢复的术语。

如果考虑稳定性，什么条件下能保证稳定性？我们主张稳定性是一种同步状态，混沌是一种不稳定的状态。也就是说，如果节点同步（所有节点达到其奇异吸引子值），其网络就是稳定的；否则既可以是双稳态的，也可以是混沌的。保证同步的条件和保证稳定性的条件相同[⊖]。

几十年来一直使用以下两种通用的技术分析耦合系统的稳定性：拉普拉斯或 Z 变换（Lyapunov，李亚普诺夫）方法和拉普拉斯特征值（谱分析）方法。我们简单地回顾这两种方法，但是我们建议感兴趣的读者可以详细学习李亚普诺夫稳定性理论（Kuramoto，1984）（Lago-Fernandez，1999）。但是，我们将详细地描述拉普拉斯特征值方法，因为它将网络状态矩阵的最大特征值与网络的稳定性联系起来。

李亚普诺夫方法

在网络上下文中，李亚普诺夫稳定性理论的一般概念十分简单。假设状态方程控制 G 中的节点 v_i，$i=1,2,\cdots,n$，形成了一个一阶微分方程［这里 $S'(t)$ 是关于时间的 $S(t)$ 变化率，$f()$ 是一个线性函数，$S(t)$ 是 G 的动态状态向量］：

$$\begin{aligned} s'_1(t) &= f_1(s_1(t)) \\ s'_2(t) &= f_2(s_2(t)) \\ &\vdots \\ s'_n(t) &= f_n(s_n(t)) \end{aligned}$$

将此转换成向量形式，注意 $S(t)=[s_1(t),s_2(t),\cdots,s_n(t)]^{\mathrm{T}}$，我们得到在 t 时间的一个简洁的 G 网络状态的变化：

⊖ 并非普遍地，但是该定义适用于本章。

$$S' = FS(t),$$

这里 F 是一个矩阵，等同于将 $f_i()$ 应用到每个 $s_i(t)$。给定初始条件，我们可以使用 Z 变换将这个时域微分方程转换为相当的复域代数方程组：$Z\{s(t)\} = s(z) = \sum_0^\infty \{f(t)z^{-t}\}$。例如，常数的 Z 变换，$f(t) = 1$ 是无限和，$f(z) = \sum_0^\infty \{f(t)z^{-t}\} = \sum_0^\infty \{z^{-t}\}$：

$$f(z) = \sum_0^\infty \{z^{-t}\} = 1 + \sum_1^\infty \{z^{-t}\} = 1 + z^{-1}\sum_0^\infty \{z^{-t}\} = 1 + z^{-1}f(z)$$

本质上，在时域上的常数导致复数域上的转换。这个转换由 z^{-1} 表示。求解 $f(z)$，我们得出

$$f(z) = 1 + z^{-1}f(z)$$

$$(1 - z^{-1})f(z) = 1$$

$$f(z) = \frac{1}{1 - z^{-1}}$$

注意只有当 $|z| < 1$ 时结果才有效。如果 $|z| = 1$，表达式 $f(z)$ 失效。复平面上的奇异点被称为极。因此，$|z| = 1$ 是这种特殊变化的一个极。按照李亚普诺夫稳定理论，系统的状态方程在两极失效。所以，有必要避免在两极求解 z 平面方程上的解。

Z 变换理论被应用到有限差分方程而不是微分方程。因此，我们可以通过将 $s'(t)$ 替换为 $\Delta s(t)$，将 Z 变换理论应用到一般状态方程上。一阶差分 Z 变换，$\Delta s(t) = s(t+1) - s(t)$ 是

$$Z\{s(t+1) - s(t)\} = z^{-1}s(z) - s(0) - Z\{s(t)\} = (z^{-1} - 1)s(z) - s(0)$$

考虑简单差分方程的解，$\Delta s(t) = h(t)$，这里 $h(t)$ 是 t 的线性函数。我们将时域差分方程转换为复平面中的更简单的代数方程，然后求解 $s(z)$。我们将 $s(z)$ 转换为时域，或者通过注意两极在何处研究其稳定性。对于一般方程，这里给定 $\Delta s(t) = h(t)$，我们得到

$$(z^{-1} - 1)s(z) - s(0) = Z\{h(t)\} = h(z)$$

$$s(z) = \frac{h(z) + s(0)}{z^{-1} - 1}$$

既然我们主要对网络的稳定性而非最终状态感兴趣，我们可以选择评估 $\Delta s(t)$ 随着 t 无限地增加的行为，或者在复平面 z 上接近两极时的 $s(z)$ ⊖。例如，$s(z) = z/(z-1)$ 在 $z = 1$ 时有一个极：

$$s(z) = \frac{z}{(z-3)(z+2j)}$$

其中，2j 是一个虚数，具有两极，分别位于 $z = 3$ 和 $z = -2j$。

李雅普诺夫稳定性理论认为，当且仅当复域解的极位于或者在单位圆之内时，线性系统是稳定的。换句话说，如果解中的所有极位于由 $|z| \leqslant 1$ 定义的单位圆时，我们就可以保证线性系统不会失效。按照稳定网络来讲，我们说明复域解也服从李雅普诺夫定理。但是，正如我们将要看到的，这不意味着振荡器网络将会同步，这只是一个开始。

谱分解

谱分解是网络的系统矩阵转化为特征值的过程，然后分析最大的特征值确定收敛到一个吸引子。一般而言，在时域解中特征值小于 1 和一阶时域微分中小于零的特征值导致稳定。不过，我们会举几个例子，说明这些条件还不足以保证网络稳定。满足上述要求的特征值是必要的，但不是保证网络稳定性的充分条件。

⊖ 极是复平面中的奇异点，例如，函数 $s(z)$ 在该点是无限的。

$S(t) = F^t S(0)$ 在时域中解收敛可能意味着达到一个奇异吸引子，以便随着 t 无限增长使得 $|S(t+1) - S(t)| = 0$。或者，收敛意味着导数随着 t 的增长而减少，随着 t 无限增长时，$|S'(t)|$ 趋近于0。在这两种情况下，实际的解 $s(t)$ 可能会达到零，也可能不会达到零。事实上，如果网络同步，因为所有的节点达到相同的非零奇异吸引子，它可能发生。因此，我们有两个相似但不完全相同的同步标准：（1）网络的所有节点的状态可能会到达零，并保持下去；（2）所有网络节点的状态可以到达相同的非零值，并保持在这个值。

当然，网络可能不同步，在这种情况下节点的状态可能会失效，或在两个或更多的吸引子中振荡。例如，两个节点由一条单链路连接的简单的杠铃形网络的状态方程给出如下：

$$s'_1(t) = -s_2(t); s_1(0) = 1$$
$$s'_2(t) = -s_1(t); s_2(0) = 0$$

矩阵形式表示为：

$$\boldsymbol{S}'(t) = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix} \boldsymbol{S}(0)$$

矩阵的谱分解产生两个特征值，1 和（-1）。最大的特征值是 1，所以我们怀疑该网络不能抑制不稳定的状态迁移。其实，杠铃形网络是最低限度的不稳定，会在两个吸引子之间振荡。更具体地说，杠铃形网络是一个带有混沌映射的双稳态振荡器，如图 9-1b 所示。

正如我们在本章稍后看到的那样，谱分解的问题在于这些特征值没有足够大的负值以抑制 G 上的所有混沌轨迹（Wang，2002a）。网络的拉普拉斯矩阵中的最大非平凡特征值是谱隙 $\sigma(G)$，它代表由邻接矩阵表示的线性系统的抑制最少量。在某些情况下，我们可以使用谱隙来确定网络是否同步，但问题在于书写本书的时候关于稳定一个特定的网络需要多大的网络谱隙负值还没有答案[⊖]。

网络同步问题的通解仍然是一个开放研究问题。相反，我们遵循以下保守一些的做法，通过谱分解和图论来研究每个网络。我们利用谱分解获取必要但不充分的判断标准，利用各种图论技术获得必要的额外条件，以保证同步：

1. 首先，我们检测极其简单的鸣叫蟋蟀社会网络，并注意到每当社会网络中至少包含一个奇数长度的回路（例如，一个大小为 3 的三角子网）时，鸣叫网络便成为双稳态。蟋蟀网络中的每个节点只为两种状态之一：如果个体监听，每个节点的值是 0（白色）；如果个体鸣叫，则为 1（红色）。

2. 接下来，我们将蟋蟀网络推广到由 Atay 等人提出的模型网络，其中每个节点的状态采取一个数值，同步算法也就是节点与其邻接节点之差的平均值。该网络具有特殊性，因为系统状态方程是网络的拉普拉斯矩阵的函数。因此，我们推测可以利用谱隙的方法进行分析。但是，我们知道这并不如预期的那样成功，因为我们不知道必须减少多少网络的谱隙来抑制所有混乱的轨迹。

我们接下来转向研究更普遍而且也许更有用的由作者提出的网络类。我们尝试稳定于基尔霍夫第一定律的网络建模，即流入节点的商品总和必须等于流出节点的商品总和。该模型很有用，因为它能表示很多真实世界的系统，如电子电路、电网等。我们发现当两个条件得到满足时，基尔霍夫网络同步：（1）基尔霍夫特征值小于 1；（2）其系统矩阵 $\boldsymbol{B}$ 的最大元素上升到 n 次方，也小于 1。

最后，我们研究了虚拟的 Flatland Pointville 社会网络，假设市场营销人员既扩散产品的正面也扩散负面的促销（buzz）。Pointville 的 buzz 网络是口碑传播网络。营销人员在一个节点播种一

⊖ Wang 的理论（Wang，2002a）认为，$\delta \leqslant (-T/c)$，网络同步，但是 T 没有定义。

个正的种子，在另一节点播种负的种子。假设 Pointville 社会网络是一个封闭的口碑传播网络，像传染病一样正的种子和负的种子在网络中传播。节点受到与生俱来的固执和邻居的平均意见的影响。我们断定最大的影响，不管是正面的还是负面的，是由接种网络的 hub 节点来实现的。如果一个营销人员要在竞争中获胜，最好的办法就是确定并（用市场营销促销来）接种网络的 hub。

buzz 网络始终同步，但本章最后一节回答的问题是："一个促销者对整个网络有什么样的影响?" 我们说明整个网络同步带有偏向朝向由网络的 hub 所发起的促销。如果 hub 是正的，整个网络最终以正的偏向终止；如果是负的，整个网络会以负的偏向终止。然而，当存在多个带有同样度的 hub 时，这种规则则例外。

9.2 蟋蟀社会网络

考虑一个简单的由蟋蟀作为成员（节点），在听得见的范围内与另一只邻居蟋蟀链路组成的社会网络[⊖]。蟋蟀在同一时间间隔内既可以监听也可以鸣叫。因此这个网络的每一个节点都处在两个状态中的一个：监听由白色表示，鸣叫则由红色表示。在每次鸣叫之后，蟋蟀至少监听一个时间间隔，然后再次监听或鸣叫。只有当它们听到来自另一只或是更多的蟋蟀（邻接节点）的鸣叫后，它们才会鸣叫。最初需要选择一只蟋蟀鸣叫。

下列 Java 编码实现上述简单的监听 - 鸣叫模型。每个节点被涂成白色（监听）或红色（鸣叫），并分配一个 next_state 值，鸣叫就像传染一样从一个邻居传到另一个。最初只有一个节点是红色的。当模拟开始时，这个节点影响到它最近的邻节点。一个被传染了的节点就会在下一时间步变成红色，将 next_state 值设置为 2。next_state 值被减 1，可能是 0。当它变为 1 时，这个节点变成红色，否则涂成白色。然而取决于网络拓扑，计时器可能是 1 或 0，或者在变为 0 之前设置为 2。以这种方式，节点在红色或白色间交替，同时试着与邻节点同步。

这种鸣叫算法实现起来非常简单，但是很微妙。方法 NW_doChirp() 经过每个节点两趟。第 1 趟确定是否每个节点都有一个邻接节点在当前时段内鸣叫，第 2 趟更新每个节点的状态。

第 1 趟：网络中的每个节点，确定是否有一个邻接节点正在鸣叫（红色），如果是的话，就安排节点在将来的时间步鸣叫（在红色和白色间交替）。这是通过将 next_state 设置为 2 完成的，因为在第 2 趟中，在测试 next_state 是 0 或 1 之前，next_state 就会减 1。

第 2 趟：如果 next_state 是 1，网络中每个节点涂成红色；否则就涂成白色。无论如何节点的 next_state 减 1，因此蟋蟀在两次鸣叫之间可以休息一下。

```
public void NW_doChirp(){

    Shuffler random_list = new Shuffler(nNodes);
    for(int i = 0; i < nNodes; i++) random_list.card[ i] = i;
    random_list.SH_Shuffle(nNodes);      //Scramble processing order
    for(int n = 0; n < nNodes; n++) {    //Pass 1: Find Red Neighbors
     int i = random_list.card[ n];       //Process nodes at random
     for(int j = 0; j < nLinks; j++){    //Find neighbors
       if(Link[ j].head == i && node[ Link[ j].tail].color == Color.red
               ||
       Link[j].tail == i && node[Link[j].head].color == Color.red){
        node[i].next_state = 2;          //Future Red
```

⊖ 假定蟋蟀有耳朵!

```
    }
   }
}
for(int i = 0; i < nNodes; i++){    //Pass 2: Listen or Chirp
 node[i].color = Color.white;       //Listen
 node[i].value = 0;                 //For chaotic map
 if(node[ i].next_state == 1) {     //Alternate back-and-forth
 node[i].color = Color.red;         //Chirp
 node[i].value = 1;                 //For chaoticmap
}
node[i].next_state- -;                //Flip-Flop state
if(node[ i].next_state < 0)
 node[i].next_state = 0;           //Bounded: 0: white, 1: red
}
}//NW_doChirp
```

这段程序代码使用来自类 Shuffler 中的 SH_Shuffle 方法，就像我们之前做过的很多次一样，随机化节点的处理顺序，以便保证仿真模拟没有引入偏差。它也将每个节点值设置为 1 或 0，对应于红、白色，程序 Network. jar 的混沌映射函数能够显示任何个体的状态轨迹。

网络中这种算法的模拟既可以产生如图 9-1b 中所示的双稳态的混沌映射，也可以产生如图 9-1c 中所示的混沌映射。因此，这个网络永远不会同步，相反当某些条件满足时达到双稳态状态（监听 - 鸣叫 - 监听）。

9.2.1 蟋蟀社会网络的同步性质

蟋蟀是在想要鸣叫的时候便自由开始鸣叫的忠实个体，但是一旦它们开始，它们也会刺激相邻的蟋蟀鸣叫。这也就引起了一种反馈效应，而附近的蟋蟀鸣叫回应——导致鸣叫复发。与传染病不同，邻伴开始鸣叫时它就中断。随后蟋蟀们都开始鸣叫，传染到整个网络。

我们在前面章节中学习过，传染病要么消失，要么持续。当它们持续下去时，传染病在不同拓扑的网络中就有不同的行为，主要是因为网络的谱半径不同。传染密度随着时间的推移而波动，它们经常会显示出混沌行为。蟋蟀社会网络显示出类似于“鸣叫传染病”的行为吗？鸣叫在网络中的传播像传染病吗？假设一只蟋蟀随机地开始鸣叫，这将导致其他蟋蟀的鸣叫等，这种连锁反应会停止吗？

我们从自然界了解到某些品种的蟋蟀、萤火虫同步它们的鸣叫和闪烁——也就从一次随机的鸣叫开始，某一品种的蟋蟀会在某个时刻开始一致鸣叫。经过一段不和谐，就会涌现有节奏的同步鸣叫。我们断定这种涌现的行为自发的在某些网络中发生，但不会在别的网络中发生。蟋蟀社会网络的拓扑结构决定它是否同步。但是与早期小世界网络中的同步研究相反，这里的蟋蟀网络中的同步与小世界无关，而是与小世界中的回路次序有关（Watts，1998，1999a）。

蟋蟀社会网络在什么条件下同步呢？正如我们将要看到的，这个简单的控制问题有一个复杂的解。图 9-2 中使用由八只蟋蟀组成的小的随机社会网络，演示了鸣叫蟋蟀控制问题的复杂性。图 9-2a 中显示了一个由 $n = 8$ 只蟋蟀和 $m = 10$ 条关系组成的初始网络。图 9-1b 和图 9-1c 显示了经过一段混沌后会发生什么——鸣叫在两半个网络中切换，一个重复的模式涌现——当其中一半鸣叫时，另一半监听，反之亦然。以持续有节奏的模式反复鸣叫——但没有同步涌现。

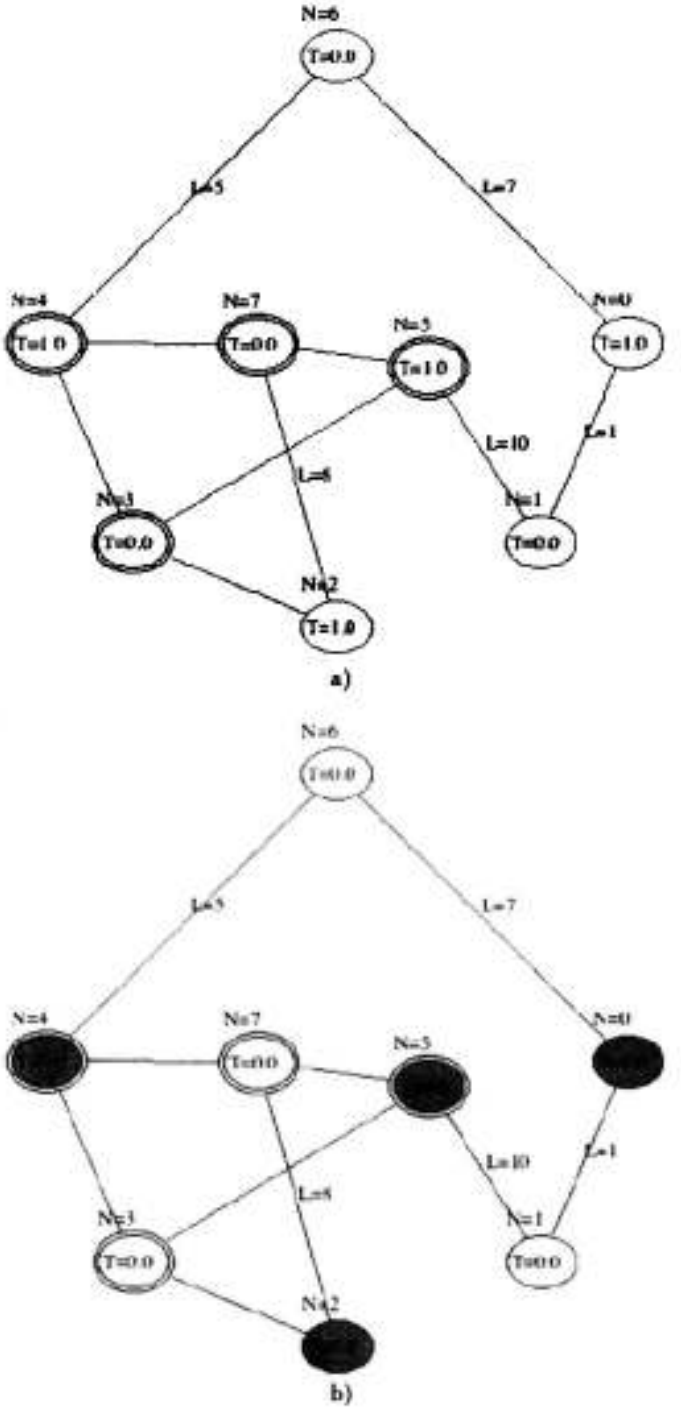


图 9-2 一个随机蟋蟀社会网络：a）原始网络（$n = 8$）；b）带阴影（红色）节点显示同时鸣叫；c）交替鸣叫节点，而其他节点监听；d）同步鸣叫（已经添加了一条链路）。在 b）和 c）中，鸣叫是不同步的，并在其中一半和另外一半网络间交替；在 d）中，鸣叫通过添加单独链路而同步——蟋蟀一起鸣叫，一起监听

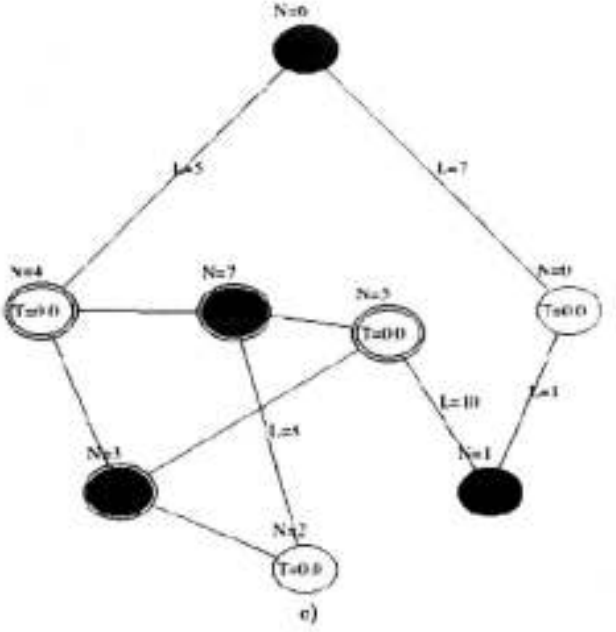


图 9-2 （续）

图 9-2d 与图 9-2a 中最初的网络是相同的，除了一个例外——在节点 0 和 5 之间插入一条链路。现在网络同步而与具体哪个成员开始鸣叫无关。经过简短的传播之后，鸣叫到达所有节点，蟋蟀开始一起鸣叫，在一起发声和一起监听中替换。稍加修改后的网络就会同步！为什么呢？

考虑图 9-3a 中的环形网络，包括 $n=8$（偶数）个节点。选择任意初始节点导致 50% 的节点一起鸣叫，其他的 50% 监听。不考虑鸣叫从哪里开始，涌现 50% 监听而另外 50% 鸣叫的替换模式。现在增加一条链路，图 9-3b 中环形网络平分为 2 个回路子图。50% －50% 替换模式涌现，因为由两部分形成的两条回路的长度是偶数。

接下来，在图 9-3c 中增加了一条链路，以便于两条回路的长度是一个奇数——在这种情况下，1 个是 5 跳长度而另一个是 7 跳长度。这个网络很快就同步了，因为是奇数回路。的确，这

个现象会在所有网络（对于蟋蟀鸣叫网络）中发生。

鸣叫蟋蟀社会网络是双稳态的——无论在随机网络还是环形网络中，大约50%同步。个体同步的百分率取决于网络的拓扑（参见表9-1）。尽管鸣叫网络不会100%同步——也就是所有节点的状态永远不会在同一时刻到达一个奇异吸引子。但是，如果网络包含一条奇数长度回路，所有节点会在两个吸引子之间振动——在状态空间中，一个在（1，0），另外一个在（0，1）。一旦发生这种情况下，网络就会无限地振荡下去。

表9-1总结了本书研究的结构化的和非结构化的鸣叫蟋蟀的仿真结果。在每个例子中，监听－鸣叫－监听涌现模式会在节点子集之间转换或者最终取得以锁定步调方式对网络中所有节点进行双稳态振荡。包含不同的奇数长度的回路子图的网络行为与具有偶数个节点或在回路中具有偶数个节点的网络行为是不同的。回顾由节点 $v_1, v_2, v_3, \cdots, v_n$ 返回到 v_1 的路径形成的回路，回路的长度等于其中包含的节点个数。

注意包含奇数长度回路的网络大部分会变成双稳态振荡（触发器），而其他的则不会。换句话来讲，如果 n 是奇数或者回路子图至少包含一个奇数节点，那么整个鸣叫网络会振荡。在一个偶数网络上增加一条链路而足以使整个网络变成双稳态！例如，当 n 是奇数或至少回路子图包含奇数个节点时，表9-1中的所有网络的鸣叫就会变成双稳态，否则不会。

当 n 是奇数时，简单的蟋蟀网络变成双稳态，当 n 是偶数时就不是，这个原因是很显然的。在一个偶数网络（或回路）中，红－白节点模式会在红白两色之间交替——红、白、红、白、红，一直顺着这个回路持续下去。在一个奇数网络（或回路）中，这种模式就会被两个或更多的相邻红节点中断——红、白、红、红、白等。连续的红、红节点就会导致NW_doChirp()方法重新设定两个节点的next_state为2，这打乱了它们的节奏。就如同游行乐队中的某个人以一个节奏快走两步，就会跨越其他人。因此，算法使每个节点与整个网络的节奏对齐。这最终会导致形成全局双稳态。

正如结果证明，鸣叫网络是下面要研究的更加通用的Atay网络的一个非常特殊的例子。如果它们包含了一个3节点回路，鸣叫网络将会同步——一个保证同步的三角回路，或者至少一个蟋蟀主导着鸣叫，通过坚定地坚持鸣叫，严格地按照鸣叫－监听－鸣叫－监听－鸣叫……顺序。因此，在如鸣叫蟋蟀的生物系统中的同步，有两种解释：（1）一个增强同步的网络拓扑结构；（2）一个占主导和坚持不间断节奏行为的独立成员。

该结果与曾激励网络同步研究的先驱们的研究结果相悖（Watts，1998，1999a；Hong，2002；Barahona，2002；Atay，2006）。流行的观念是Atay（Atay，2006）和其他人提出的小世界拓扑结构保证同步显示是错误的，但是这里给出的分析是早期研究，说明如何固定住并增加稳定控制子网实际上能够稳定很多网络——但是不是所有！鸣叫只是很多导致同步行为中的一个。

表9-1 有规则和无规则网络的同步结果

网络	双稳态?	备注
杠铃形	不是	双稳态振荡
线形	不是	50% -50%振荡
2－规则	是	奇数回路
完全	是	奇数回路
超环形	是	奇数回路
二叉树	不是	没有回路
超立方	不是	偶数回路
随机	几乎是	近似奇数回路
小世界	是	有聚类
无标度	几乎是	近似奇数回路
星形	不是	没有回路
环形	N = 奇数	奇数回路

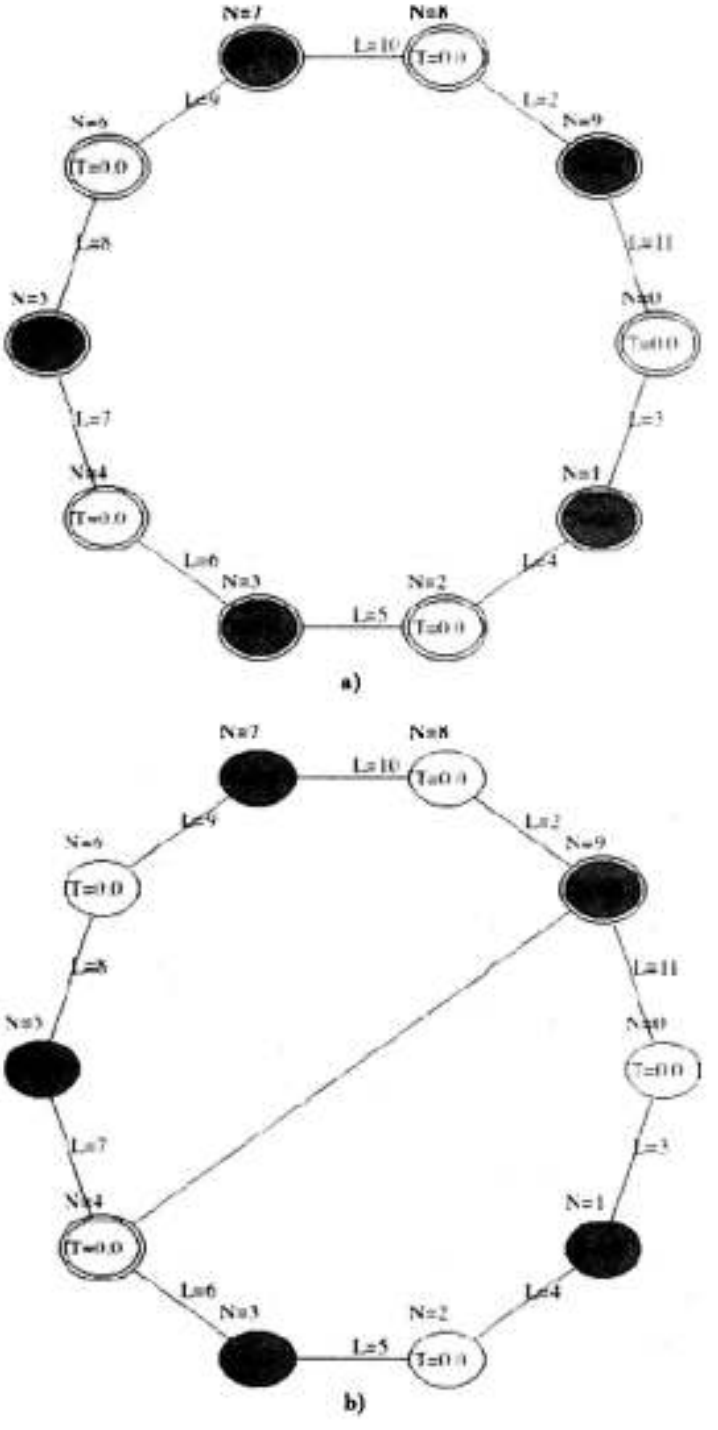
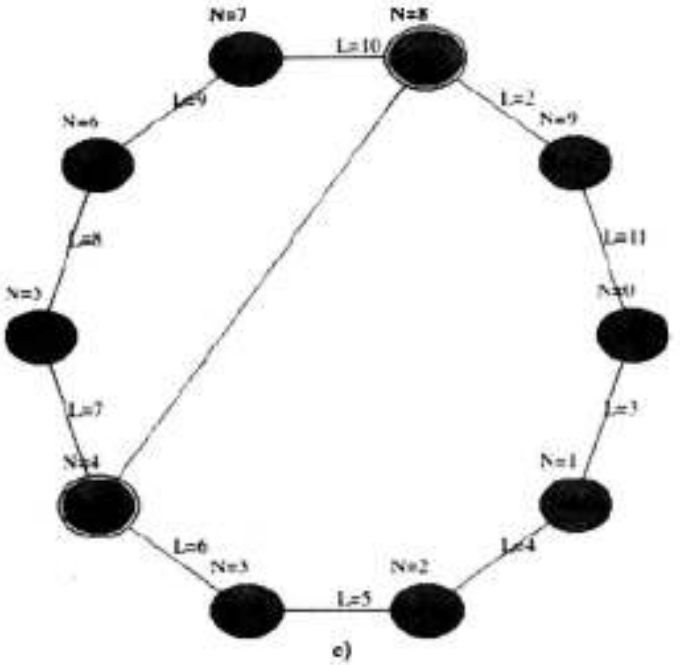


图 9-3 鸣叫环形网络：a）偶数环形网络不同步（$n=8$）；b）带有两个 $n=6$ 的子图的偶数环形网络；c）带有两个奇数子图的偶数环形网络，一个 $n=5$，一个 $n=7$。只有带有奇数回路的网络是双稳态鸣叫



c)

图9-3 （续）

9.2.2 更加通用的模型：Atay 网络

图9-3和蟋蟀鸣叫算法阐明了两个一般的同步原则：（1）网络可能因为拓扑而同步或不能同步；（2）也可能因为状态变化算法的性质而同步或不同步。拓扑决定了可能会阻碍同步或不会阻碍同步的反馈信号，状态变化算法的性质决定了反馈信号的幅度。一个反馈幅度可能会导致一个网络“失效”，特别是如果随着时间的推移而无限地增加时更是如此。因此，我们既要考虑算法，又要考虑网络拓扑，这样才能建立起网络同步的必要条件。

首先，我们用 Atay、Biyikoglu 等人建议的状态方程归纳出状态变化算法（Atay，2006），并且说明网络拓扑在决定节点同步（所有节点值趋向同一值）或不同步（节点值振荡或“失效”）时扮演一个关键的角色。然后，我们建立有关网络拓扑和节点值导致稳定的一些必要的条件。我们得到的结果类似于上一节但是更普遍的状态方程表示。此外，我们说明在许多非小世界网络，与前面研究的小世界网络相反，同步是可能的。

下面研究的 Atay 算法在有些网络中同步而有些则不同步（参见表9-2）。例如，线形、星形、超环形、二叉树、超立方和其他多数规则网络中，不能实现 Atay 同步，但是在 2－规则、完全、无标度网络中却能同步。在确定的条件下，在随机、小世界和环形网络上算法成功。更进一步来讲，同步的能力与网络谱隙无关，如表9-2所示。问题是 Atay 算法为什么在一些网络中同步，在另一些网络中则不同步。

设节点 i 的状态由 s_i 给出，因此列向量 $S(t)$ 就是 G 网络在 t 时刻的系统状态。$S(t)$ 的时间变化率表示为 $S'(t)$，转置矩阵是 $S^{\mathrm{T}}(t)$，$f(S(t))$ 是关于 $S(t)$ 的线性函数。将 $f()$ 作为输出函数，将状态 $S(t)$ 转换成在每一个时间步 t 的值。我们定义动态网络系统的系统状态如下：

$$S(t) = [s_1, s_2, s_3 \cdots, s_n]^{\mathrm{T}} = G \text{ 的状态向量；} S_j(t) = S(t) \text{ 的 } j \text{ 行}$$

$$f(s_j) = \text{节点 } j \text{ 的线性状态函数；} f(S) = \text{应用于系统的函数}$$

Atay 状态方程 $S'(t) = f(S(t)) + \sum_{j=1} |f(S_j(t)) - f(S_i(t))|$，这里 $\sum_{j=1}$ 是节点 i 的邻居总和，即 $i \sim j$。

我们想要知道在什么条件下状态方程应用到 G 网络中会导致同步。换句话说，$S'(t)$ 随着 t 的无限增长会趋向于0吗？用更正式的术语，我们讲由 G 网络和状态方程 $S'(t)$ 构成的系统 $H(t) = \{G(t), S(t)\}$，具有：

1. 如果 $S'(t)$ 趋向于0，则稳定。例如，如果所有节点的状态趋近于0或者保持常量，就会保持不变。

2. 如果 $S(t)$ 在一定的范围值内振动，则瞬态。如果它的节点值在两个吸引子中振荡，则瞬态的网络保持双稳态。

3. 如果 $S(t)$ 的任何元素无限，则不稳定或失效。例如，如果一个或更多的节点值变成不断增加的正数或负数，最后达到无限大或负的无限大，那么我们就说网络失效了。

Atay 等人（Atay, 2006）提出下列状态方程，这里我们加以修改并深入探讨。Atay 网络是连通的网络，其中每个节点的变化率由每个节点和它邻节点间的平均差来决定（这里 d_i = 节点 i 的度）：

$$S'(t) = \sum_{j \sim i} \frac{f(S_j(t)) - f(S_i(t))}{d_i} = \sum_{j \sim i} \frac{S_j(t) - S_i(t)}{d_i}$$

表 9-2 网络中的 Atay 同步（n=100）

网络	同步（是/不是）	典型谱隙
线形	不是	-0.001
星形	不是	-1.000
2-规则	是	-0.020
完全	是	-100.0
超环形	是	-0.382
二叉树	不是	-0.018
超立方	不是	-2.000
随机	大部分	-0.287
小世界	大部分	-0.080
无标度	是	-0.615
环形	有时候	-0.004

Atay 等人选择 $f()$ 恒等函数将状态映射成一个输出值和离散差分方程。在有限差分形式，它是（$j \sim i$ 指的是 i 的邻接节点）：

$$s_i(t+1) = s_i(t) + \sum_{j \sim i} \frac{s_j(t) - s_i(t)}{d_i}$$

对每个节点 i，这个状态方程计算 next_state，$s_i(t+1)$ 通过增加存储在每个节点的当前值 $s_i(t)$，增加量为每个节点当前值和它邻接节点之差的总和，并且被每个节点的度标准化。既然每个节点有 d_i 个相邻节点，它与所有邻节点的平均差值相等。

很清楚，如果差值总和减小到0，在每个节点上的差值，$|s_i(t+1) - s_i(t)| = 0$，网络稳定在某个公共值左右。差值振荡，意味着网络是瞬态的。或者在某些情况下，因为状态值无限地增长或减少，网络就会失效。这三种情况中哪种拟合 Atay 公式？

下列 Java 编码实现离散的 Atay 算法。方法 NW_doAtay() 假定网络上所有节点的初值存储在 node[i]. next_state。程序 Network. jar 为每个节点分配一个在 2 到 10 之间的随机值，但是这是相当任意的。然后，它以两趟将 Atay 状态方程应用到所有节点中：第 1 趟更新每个节点值为它的下一个状态，第 2 趟对每个节点以离散的形式应用于下一个状态方程。结果在第 1 趟中的下一时间步，next_state 更新网络。因此在对应混沌映射中的轨迹是特定节点的 next_state 的所在地所对

应的当前 value（参见图 9-4）。

Java 方法随机处理节点以避免仿真偏差的引入。next_state 的初始值是随机分配的，算法无限地运行。在这种情况下，同步意味着所有节点最后达到同一个值。这对应于所有节点的同一个奇异吸引子。

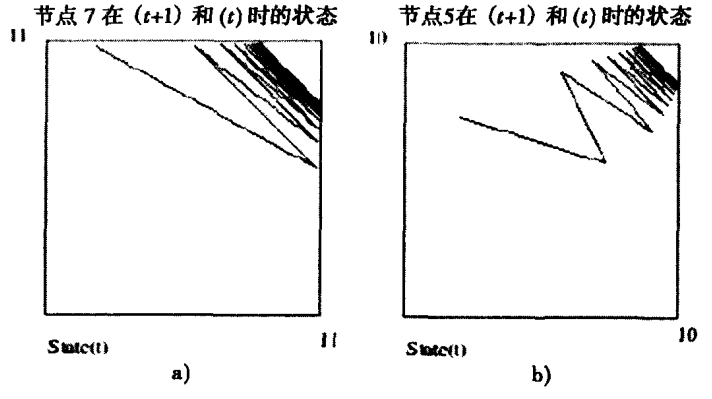


图 9-4 Atay 网络的混沌映射轨迹：a）稳定节点在网络同步时达到奇异吸引子；b）网络不同步时，双稳态节点在两个吸引子之间振荡

```
public void NW_doAtay(){
       for(int i = 0; i < nNodes; i++){        //Pass 1: Update state
         node[i].value = node[i].next_state;
        }
       Shuffler random_list = new Shuffler(nNodes);
       for(int i = 0; i < nNodes; i++) random_list.card[ i] = i;
       random_list.SH_Shuffle(nNodes);         //Scramble node order
       for(int n = 0; n < nNodes; n++){        //Pass 2: Next State
         int i = random_list.card[ n] ;        //Random selection
         node[i].next_state = node[i].value;//Right-hand-side
         for(int j = 0; j < nLinks; j++){    //Neighbors
           int left = Link[ j] .tail;
           int right = Link[ j] .head;
           double d = node[ i] .degree;        //Degree of node
           if(left == i) {                     //f(si)+sum[ f(sj)-f(si)]
                node[i].next_state += (node[right].value -
                                       node[i].value)/d;
           }
           else if(right == i) {
                node[i].next_state += (node[left].value -
                                        node[i].value)/d;
           }
         }
       }
    }//NW_doAtay
```

9.2.3 Atay 网络的稳定性

在大部分情况下，Atay 网络既可以通过达到奇异吸引子而同步，也可以在双稳态值之间振荡。因为度 d_i 充当减振因子，Atay 网络就不会“失效”。但是究竟什么决定 Atay 网络是否到达瞬态的双稳态或完全同步状态？

表9-2是当 $n = 100$ 时对各种网络通过运行 NW_doAtay( )获得的，并观察到汇聚到稳定固定点或节点振荡值随着时间而变化。很清楚，某些网络同步，某些则不同步。更重要的是，Atay网络的谱隙与同步能力之间没有关系[⊖]。但是我们可以使用一个简单的方法保证同步。

我们证明了网络必须至少包含一个固定节点或至少一个三角回路才能同步。如果一个节点其值是确定的则被认为是固定住的。固定住的节点一直处于初始状态（或者由外部控制建立的一些状态）。三角回路是长度为3的回路，由3个节点相互连接构成。它是最简单的聚类，聚类系数为1.0。

现在我们可以把Atay网络的同步能力与其拓扑关联起来了。在表9-2中列出的所有网络至少包含一个三角回路同步，而所有其他的振荡，即稳定的Atay网络中的节点到达一个公共值，而在瞬态Atay网络中的节点值振荡——通常在2个吸引子之间交替。

以下策略过程用以解释为什么当Atay网络包含一个三角回路时就同步，否则就不同步：

1. 证明Atay杠铃形网络（$n=2$，$m=1$）是瞬态双稳态振荡（触发器）。

2. 证明Atay三角形网络（$n=3$，$m=3$）是稳态网络。

3. 证明当 n 是奇数时，Atay环形网络是稳态的，当 n 是偶数时，Atay环形网络是瞬态的。

4. 证明如果增加链路构成奇数长度回路时，增加随机链路的Atay环形网络是稳态的，否则是瞬态的。

5. 对环形网络应用约化代数以决定Atay环形网络是否满足稳态的必要条件。

6. 概括1～5步骤中获得的结果用来解释为什么小世界同步要比随机或规则网络同步更容易发生。特别注意，不是所有的小世界都同步！

7. 进一步总结，我们认为，但没有得到证明，至少有一个节点被固定住时，Atay网络同步。这从观察中得出固定节点类似于稳定三角回路。

8. 得出这样的结论：固定的（算法的一个性质）、三角回路（网络拓扑性质）就足以引起Atay网络的同步——与前面的结果相反。

首先，我们将Atay算法应用到图9-5a中的杠铃形网络，进行更完整的瞬态行为分析：

$$s_1(t+1) = s_1(t) + \{s_2(t) - s_1(t)\} = s_2(t)$$
$$s_2(t+1) = s_2(t) + \{s_1(t) - s_2(t)\} = s_1(t)$$

或者，在矩阵中，$\boldsymbol{A}$ 是杠铃形网络的邻接矩阵：

$$S(t+1) = \boldsymbol{A}S(t)\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix} = S(t)$$

解初始状态矢量 $S(0)$，得到

$$S(t) = \boldsymbol{A}^{\mathrm{T}}S(0)$$

矩阵 $\boldsymbol{A}$ 非常简单，$\boldsymbol{A}'$ 更简单：当 t 是偶数（无变化）时，$\boldsymbol{A}' = \boldsymbol{I}$；当 t 是奇数（触发器）时 $\boldsymbol{A}' = -\boldsymbol{I}$。因此，Atay杠铃形网络是双稳态振荡：当 t 是奇数时，$S(t) = -\boldsymbol{I}S(0)$，否则就等于 $S(0)$。

现在考虑图9-5*b*的三角网络，它是稳定的。沿用相同的方法，状态方程的解区别仅在于，实际上三角网络的所有节点拥有度为2。因此，我们除以2：

$$S(t+1) = 0.5\boldsymbol{A}S(t)$$
$$S(t) = (0.5\boldsymbol{A})^{t}S(0)\text{；给定 }S(0)$$
$$\boldsymbol{A} = \begin{pmatrix}0 & 1 & 1\\ 1 & 0 & 1\\ 1 & 1 & 0\end{pmatrix}$$

⊖ 谱分解是分析中的偏爱方法，但是一般来讲它不能告诉我们是否谱隙足够大就能同步一个任意网络。

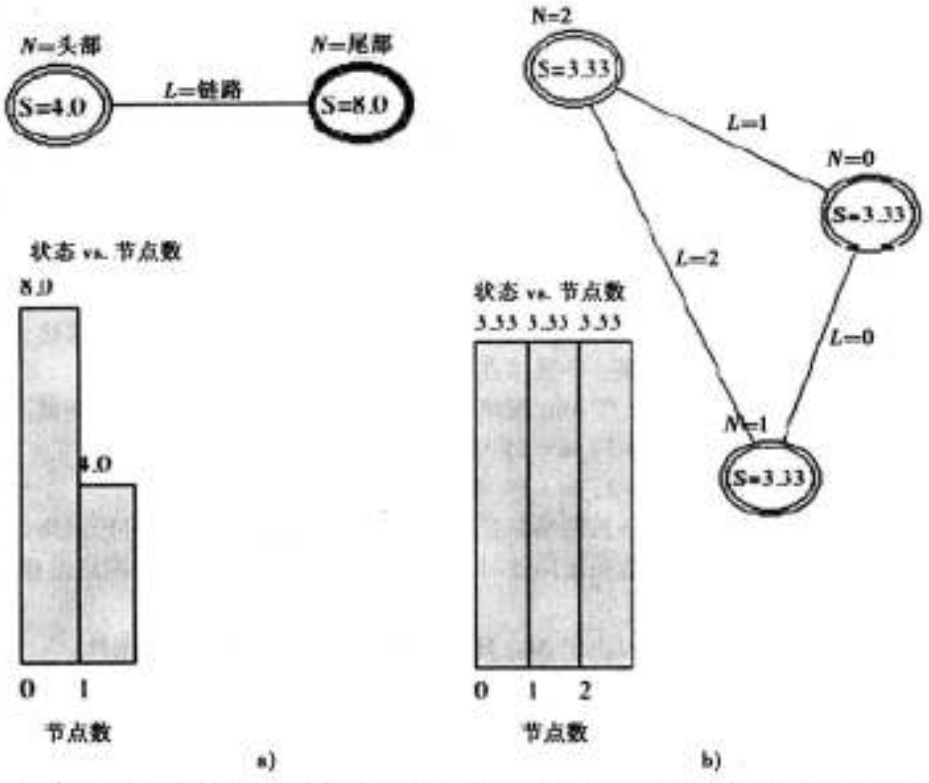


图 9-5 原始的 Atay 网络：a）由杠铃形网络实现的双稳态振荡（触发器）；b）由大小为 $n=3$ 的完全网络实现的稳定三角回路

表 9-3 环形同步的约化代数

序号	结构	约化	备注
1	固定节点	菱形	固定住节点是稳定节点（菱形）
2	奇数长度回路	菱形	稳定节点替换回路（菱形）
3	偶数长度回路	节点	单振荡节点替换回路
4	稳定－稳定对	菱形	单稳态节点替换（菱形）
5	稳定－节点对	菱形	单稳态节点替换（菱形）

注意 A 是三角网络的邻接矩阵。因此，A 的谱分解与找到的谱半径 $\rho = 2$ 是一致的，因此 $S(t)$ 随着 t 无限增长的极限为常数。

$$S(t) \to S(0)\ ,\ \text{因为}\ ((0.5)(2.0))' = 1.0$$

Atay 三角网络是稳定的。事实上，稳定的三角网络中的每一个节点的值等于初始值的平均数。不考虑三角回路中任何一个单节点的变化，Atay 三角网络被吸引到节点初始值的平均值。该网络中混沌映射上的节点轨迹看上去像图 9-4a 中的映射。

现在我们有两个构建模块网络，一个在其初始值振荡，而另一个达到等于初始节点值的平均数的稳定的奇异吸引子。如果我们将这些元素结合会发生什么？把一个不稳定回路和稳定回路结合起来，会产生组合同步。

一般来讲，在不稳定的网络中引入稳定的节点或回路会导致它变稳定。看起来稳定性在网络中像传染病一样传播。表 9-3 总结了由菱形节点指定的稳定节点和由环形节点指定的振荡节点的组合。

考虑图 9-6 中的环形网络和小世界网络。我们使用由表 9-3 中的约化规则定义的简单图形代

数来分析这些环形中的每一个元素[⊖]。首先，我们用稳定的菱形节点替换所有的固定住的节点。接下来，我们用菱形替换所有的奇数长度回路，用圆形（振荡）节点替换所有的偶数长度回路。节点对像表9-3的中描述的那样递减。我们重复表9-3的约化直到整个网络已减少为一个振荡器或者稳定的菱形。

在图9-6中，两个网络是瞬态振荡器，4个是稳定的。图9-6a中的4－环形约化为振荡器，因为它是一个大的偶数长度回路。图9-6b中的5－环形约化为稳态菱形，因为它是一个大的奇数长度回路。相反，图9-6c中的二分6－环形，包含两个偶数回路，约化成两个振荡器，因此，小世界是瞬态的。

图9-6d阐述了小世界上的约化应用是同步的，因为它至少含有一个奇数长度回路。一个回路约化成菱形，其他约化成振荡器节点。应用表9-3的“稳定－节点对”规则导致第二次约化，以稳定的节点终结。即使小世界网络有偶数个节点，它也同步。

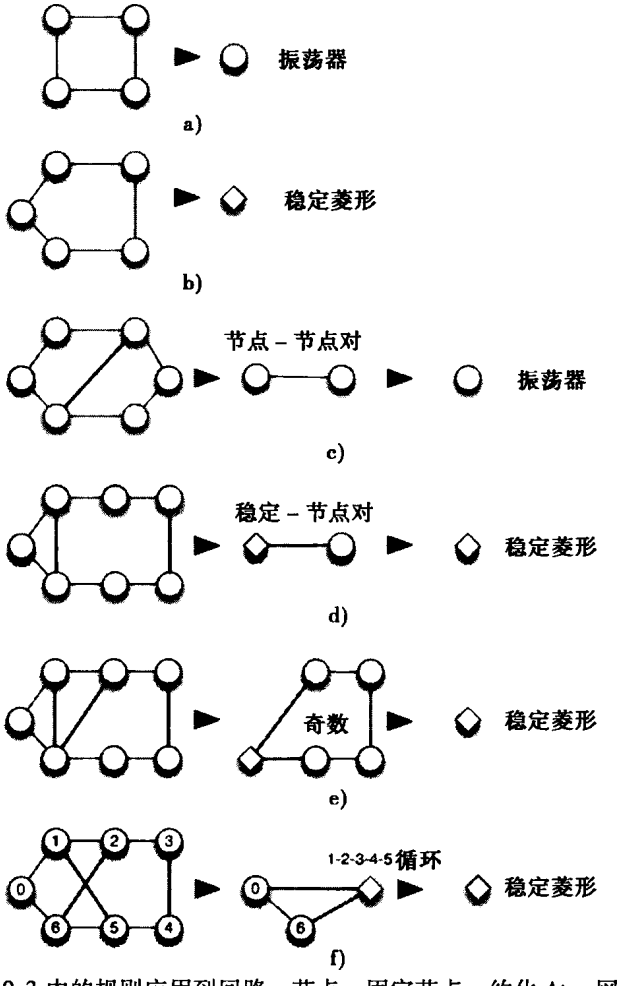


图9-6 通过将表9-3中的规则应用到回路、节点、固定节点，约化Atay网络：a）环形 $n = 4$ 是瞬态的，因为 n 是偶数；b）环形 $n = 5$ 是稳定的，因为 n 是奇数；c）二分小世界是瞬态的，因为它所有的回路是偶数；d）奇数回路的小世界是稳定的；e）嵌套回路的小世界是稳定的，因为至少一个回路是奇数；f）非嵌套回路的小世界是稳定的，因为在其约化为偶数长度回路前，是奇数回路

⊖ 为了最佳结果，自顶向下按序应用表9-3中的规则。

图9-6e阐述了约化规则应用到一个嵌套回路网络。最内部的回路是首先约化，随后是外部回路——约化包含在其他回路中的回路。在这种情况下，最内部的回路是奇数，因此它由稳定菱形替换。最外层回路保持不变，但是它是奇数，所以网络是稳定的。

因为图9-6f包含重叠的回路，它就更加有益。必须严格按照表9-3的规则约化网络。规则2必须在规则3之前应用。因此，奇数长度回路必须在偶数长度回路之前约化。将规则2应用到这种情况下，意味着将包含有1、2、3、4、5节点的回路约化到一个单独的节点。因为这是一个奇数回路，我们就用带有菱形的节点加以替代。注意，重复链路合并成一条链路，约化网络的连接矩阵符合原连接矩阵。这种小世界网络是稳定的，因为它至少含有一个奇数长度回路。

对带有固定节点的网络，我们得到类似的结果。应用规则1立即用带有稳定菱形的固定节点代替。因此，固定节点等价于奇数长度回路。

一般来讲，小世界可能不同步，但我们从前面的章节中知道小世界网络包含大量的聚类。聚类是三角回路的集合，小世界不同步的可能性很小。在大多数情况下，小世界网络同步，因为它们大多数具有回路。对于较小度的随机和无标度网络，这是正确的。

有趣的是，我们可以通过添加一个三角回路给适当的节点来同步任何Atay网络。例如，一个二叉树Atay网络将因为它的子树并不会形成回路而不同步。然而，额外添加两个节点和三条链路到它的根节点后会致使整个网络同步！令人惊讶的是，二叉树中的所有节点将汇聚到同一值。表9-4说明了这个现象，但是将研究问题的证明留给读者来完成。给定一个不同步的Atay网络，证明添加一个三角回路就可以保证它同步。

我们可以更进一步声称（未经证明），如果固定住任何一个节点，那么每个Atay网络都会同步，也就是保持任一节点的值为常数就充分保证整个网络稳定！直观地讲，这是因为单个节点的稳定性像传染病一样传播到整个（连接的）网络上。另一种解释是固定节点就像一个三角回路一样同步邻接节点。

表9-4 通过添加一个三角回路而同步的网络

网络	添加一个三角回路
杠铃形	两端中任意一端
线形	任意一个节点
二叉树	根节点
超立方	任意一个节点
随机	任意一个节点
无标度	任意一个节点
星形	任意一个节点
环形	任意一个节点

如果我们固定住一个节点而非连接一个三角回路，那么表9-4中每个Atay网络将同步。这对于2-规则、完全、小世界网络也是如此，但它们通常不需固定或添加一个三角回路就同步。这是因为它们已经包含一个三角回路。

这个结论在不同的领域具有重大意义，包括医药、物理学、计算机科学、电气工程。蟋蟀社会网络会因为一个占主导地位的蟋蟀而同步吗？人类心脏会因为一个占主导地位的神经脉冲平稳而有节奏地跳动吗？这个推测的证明留给读者作为练习。证明如果至少一个节点被固定，任何Atay网络将同步。

9.3 基尔霍夫网络

我们现在研究一类带有三角回路或固定一个节点的不同步的网络。基尔霍夫网络是任一包

含遵循基尔霍夫定律节点和链路的有向网络。这种网络建模从源节点到接收节点的商品流，是对许多实际系统的近似。但是，我们将会看到它们总的来讲要比 Atay 网络更难以同步。

基尔霍夫第一定律指出，流入一个节点的总和必须等于流出节点的总和。基尔霍夫第二定律指出，环路的电压之和必须为零。尽管这些定律控制着微处理器电路和电网的行为，它们同样为许多其他现象建模，如管道和河流的液体流动、邮政系统的信息流（忽略丢失的信件）和全球分销网络中的商品流。对稳定性问题的更一般的答案很重要，是因为它能应用到许多领域之中。

在电涌期间或一个节点损坏、关闭或离线后，基尔霍夫网络中会发生什么状况？网络能适应新的配置并且稳定下来，或者进入混沌振荡吗？这个问题对保持电力网、物流系统及交通网络的良好状态很重要。我们想知道这些系统在压力之下，甚至故障的情况下是否稳定。为了得到答案，我们再次利用仿真。

9.3.1 基尔霍夫网络模型

让我们将基尔霍夫网络定义成一个有向的网络 $K = \{N, L, f\}$，其中分别为节点 N 和链路 L 分配一个值，并且链路是有向的。每个节点都有一个*入度*（指向节点的链路数）和一个*出度*（锚定到节点的链路数）。节点的子集并没有入度链路，所以我们称之为*源端*；另一个集合没有出度链路，所以我们称之为*接收端*。我们假定所有源节点的值保持不变，而所有其他节点值都由基尔霍夫第一定律确定：

$$\text{node}[i].\text{next\_state} = \sum \text{node}[i].\text{in\_flow} - \sum \text{node}[i].\text{out\_flow}$$

其中

$$\text{node}[i].\text{in\_flow} = \sum \text{link}[j].\text{value}, \text{通过入度链路}$$

$$\text{link}[k].\text{value} = \frac{\text{node}[i].\text{value}}{\text{node}[i].\text{out\_degree}}, \text{对于锚定节点}$$

换句话来讲，每个节点的值是经进入链路的输入总和与经外出链路的输出总和之差。我们均匀地在出度链路之间分配外出流量，所以链路值是它的锚节点（尾部）的值除以锚节点的出度。这意味着 out_ flow 等于每个节点的当前值，我们后面正式地分析基尔霍夫网络的稳定性时会用到这一事实。

如果一切顺利，网络上每个节点流入和流出之差为零。如果一个节点值超过零，它是一个瞬态的状态。如果所有节点同时达到零，网络将同步——所有节点将被吸引到零固定点上。我们限制节点值大于或等于零，因为负值意味着流动反向。我们只考虑向前流动的网络。

方法 NW_diKirchhoff(int done)计算节点的下一个状态，并返回一个与节点数相等的整型值（在一时间步改变的值）。当这个数字达到零时，网络已达到某种稳定状态。否则，它会处于一个瞬态的或混沌状态。

该算法用两趟实现。第 1 趟更新每个节点的值到它的下一个状态，并将接收节点标记为红色，源节点为绿色，孤立或断开节点为黑色，其他所有节点为白色。在第 2 趟方法扫过所有的链路并通过在所有外出链路平分锚节点的值来更新每个链路的值，然后使用基尔霍夫第一定律计算每个节点的下一个状态值。所有源节点的值保持不变，所有接收节点的总和值等于通过网络流动的总的商品。

方法 NW_doKirchhoff( )返回变量 done，它是该方法已经返回零的累计次数（随网络而改变）。仅当在下一时间步没有节点值改变时，变量 stable_time 为零。变量 done 计算 stable_time 返回 0 的次数。思路是经过足够多的循环后，当网络达到稳定后就停止仿真。程序 Network. jar 使用 done 决定在网络达到稳定之前走了多少时间步。当然，这是假设每个基尔霍夫网络同步——该假设并非对所有网络有效。

正如前面所述，我们用随机处理顺序消除人为引入的偏差，最初分配随机值给节点。混沌映射中基尔霍夫网络节点的轨迹描绘了基尔霍夫第一定律在时间 $(t+1)$ 与时间 t 的对比，因此，节点的稳定吸引子位于零：

```
public int NW_doKirchhoff(int done){
        int stable_time = 0;                    //Pass 1: update nodes
        for(int i = 0; i < nNodes; i++){        //Color coding
            if(node[ i] .out_degree == 0) node[ i] .color = Color.red;
            if(node[ i] .in_degree == 0) node[ i] .color = Color.green;
            if(node[ i] .out_degree == 0
                      &&
               node[i].in_degree == 0) node[i].color = Color.black;
            if(node[ i] .value != node[ i] .next_state) stable_time++;
            node[i].value = node[i].next_state;    //Update state
        }
        if(stable_time == 0) done++;            //No state changes
        int tail_node, head_node;               //Tail and Head of Links
        for(int j = 0; j < nLinks; j++){        //Update link flow values
             tail_node = Link[j].tail;          //In case user deletes
                                                  during run
             Link[j].value = node[tail_node].value/node[tail_node].
                                                  out_degree;
        }

        Shuffler random_list=new Shuffler(nNodes);//Randomize nodes
        for(int i = 0; i < nNodes; i++) random_list:card[ i] = i;
        random_list.SH_Shuffle(nNodes);

        for(int n = 0; n < nNodes; n++){        //Pass 3: Next State
          int i = random_list.card[ n] ;        //Scrambled order
          node[i].next_state = 0;               //Next state of this node
          double inflow = 0.0;                  //Reset this in/out flow
          double outflow = 0.0;
          for(int j = 0; j < nLinks; j++){  //Across all links to this
            tail_node = Link[j].tail;
            head_node = Link[j].head;
            if(tail_node == i) {                //Flow out of this node
                 outflow += Link[j].value;
            }
            else if(head_node == i) {           //Flow into this node
                 inflow += Link[j].value;
            }
          }
    if(node[ i] .in_degree == 0) node[ i] .next_state =
                                        node[i].value;
    else node[ i] .next_state = inflow - outflow;
    if(node[ i] .next_state < 0.000001) node[ i] .next_state = 0;
  }
  return done;
}//NW_doKirchhoff
```

当且仅当其所有节点的状态同时达到零时——基尔霍夫网络奇异吸引子，基尔霍夫网络同步，即网络输入的总和等于每个节点的输出总和。因此，同步意味着所有的节点同时达到零值，并保持下去。

当 next_state 值接近零时，方法 NW_doKirchhoff( ) 对它加以修剪，以便消除奇异吸引子附近的低级别（高频率）的振荡。如果网络会同步，这加速收敛到同步——并且不会失去一般性。如果基尔霍夫网络同步，它会逐步驱使所有网络进入零状态。

9.3.2 基尔霍夫网络的稳定性

并非所有的基尔霍夫网络都是同步的，其中某些行为相当古怪。某些网络将表现出混沌，经过数百时间步也没有达到同步。其他网络只经过几次迭代后就“突然”进入同步，为什么？基尔霍夫网络稳定性分析显示，同步完全是由网络的拓扑结构决定的。具体来讲，假设网络包含长度为 A 和 B 的直接循环，如果 A 和 B 互质，那么网络将同步。网络包含长度为 A 、B 、C …的有向循环，如果 A 、B 、C …的最大公约数比 1 大就不会同步；也就是说，至少两个循环必须包含互质的跳数。我们在下面详细展开这一理论。

注意推导基尔霍夫网络同步的必要条件的策略如下：

1. 将分析的基尔霍夫网络类限制为*封闭系统*——没有源或接收节点的网络。因此基尔霍夫网络充满了反馈链路。

2. 导出基尔霍夫网络状态方程为矩阵形式，其中 $S(t)$ 是包含每个节点值的状态向量，$\boldsymbol{B}$ 是出度网络矩阵，$(\boldsymbol{B}-2\boldsymbol{I})$ 是系统状态矩阵。

3. 证明状态向量变化的时间率 $\Delta S(t)$ ，会随着 t 的无限增长而趋近于零（奇异吸引子），但是这并不足以保证同步。我们称 $(B-2I)$ 的最大非平凡特征值为基尔霍夫特征值，并将它指定为 ε_k 。

4. 证明收敛 $\Delta S(t)\rightarrow 0$ 是必要的，但并不足以保证同步，因为在循环中的*信号强度*可能会也可能不会抑制属于反馈循环的节点的 next_state 值。

5. 证明除了第一个条件 $\Delta S(t)\rightarrow 0$ 外，我们还必须保证在网络中来自循环的信号强度相互干扰（而不是加强），以便发生同步，否则网络就无限地振荡。我们证明为何基尔霍夫网络只有在包含的循环长度为相对素数时才会同步，而并非所有其他网络都会这样。

带有源节点的网络可能不会同步，这是因为所有源节点的状态保持不变。同样，带有接收节点的网络可能同步也可能不同步，这取决于其他节点的初始值。例如，线形、星形、二叉树和无标度网络可能不同步，这是因为它们至少有一个源或接收节点[⊖]。源或接收节点没有可能抑制振荡的任何反馈链路。这些网络是开放系统，因为它们至少有一个源或接收节点。

我们的研究限制为*封闭系统*——不包含源或接收节点的网络。封闭系统中的每个节点至少有 2 个度—— 一个入度和一个出度。在封闭系统中，信号流经一个节点——它们决不会“被困”在一个节点上。例如，环形、k - 规则、完全、超环形和超立方网络都是封闭系统。通过颠倒某些链路方向和重联某些链路，我们可以将随机、小世界或无标度网络转换成封闭网络。如果我们将无标度网络转换成一个封闭系统，它可能会也可能不会同步，具体取决于网络同步的必要条件。

现在假定我们进行到查找基尔霍夫网络同步的两个条件策略中的下一步。考虑图 9-7 中的环形网。这些网络是封闭系统，但其中两个同步，而另两个则不同步。我们认为图 9-7a 中的 1 - 规则环形网络和图 9-7d 中的小世界网络是同步的，而其他两个则不同步。为什么？像前面一样，

⊖ 这是部分地人工生成网络的方法——所有的链路都指向新创建的节点。

我们使用基尔霍夫第一定律，从考虑网络状态方程开始分析。

请注意，一个节点的输出总和正是在时间 t 时的节点状态值 $S(t)$。因此，基尔霍夫网络的状态方程可改写为矩阵形式如下：

$$S(t+1) = \boldsymbol{B}S(t) - S(t) = [\boldsymbol{B} - \boldsymbol{I}]S(t); \text{给定 } S(0)$$

其中 $\boldsymbol{B}$ 是出度连接矩阵，$\boldsymbol{I}$ 是单位矩阵。

出度连接矩阵为忽略所有入度连接并设置元素

$$b_{i,j} = \frac{1}{\text{out\_degree}(i)}$$

时得到，如果 $j \sim i$，其中 j 是连接节点 j 和 i 的链路的尾部节点，而 i 是头部节点。否则，$b_{i,j} = 0$。出度矩阵的行对应于头部节点，列对应于尾部节点。因此，表达式 $\boldsymbol{BS}(t)$ 代表流入，$S(t)$ 代表从每个节点的总流出。

例如，图 9-7a 中环形网络的出度连接矩阵是：

$$\boldsymbol{B} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

图 9-7b 中小世界网络的出度矩阵是：

$$\boldsymbol{B} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

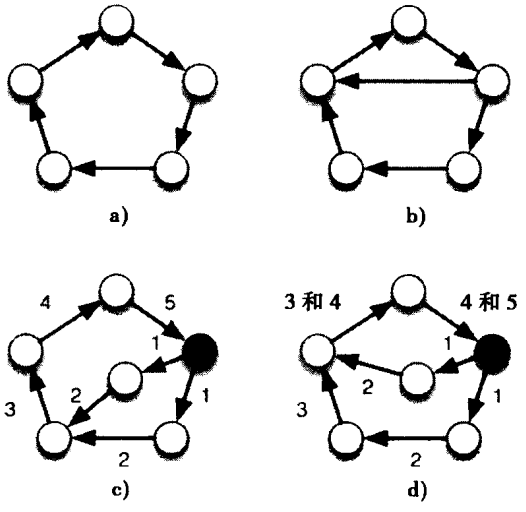


图 9-7 基尔霍夫网络：a）混沌 1 - 规则环形网络；b）和 d）稳定的混沌小世界网络；c）混沌小世界网络。网络 a）和 c）所示网络不同步，因为它们违反基尔霍夫稳定条件中的 1 个或 2 个，但是 b）和 d）是同步的，因为 $\varepsilon_k \leq 1.0$，它们包含互质的 2 个循环长度

基尔霍夫网络的必要不充分同步条件是 $\Delta S(t)$ 收敛于零。从状态方程的两边减去 $S(t)$ 并以初始状态 $S(0)$ 和出度矩阵 $\boldsymbol{B}$ 求解方程。我们得到

$$\Delta S(t+1) = [\boldsymbol{B} - 2\boldsymbol{I}]S(t); \text{给定 } S(0)$$

因此

$$S(t) = [\boldsymbol{B} - 2\boldsymbol{I}]^t S(0); t > 0$$

为使 $S(t)$ 趋近于零，很有必要使 $= [\boldsymbol{B} - 2\boldsymbol{I}]^t$ 趋近于零。$[\boldsymbol{B} - 2\boldsymbol{I}]$ 谱分解产生状态方程解的最大非平凡特征值。假如我们称这为网络的基尔霍夫特征值 ε_k。那么为了使下一个状态保持有界，基尔霍夫特征值必须是正的且小于1。如果 ε_k 是负数，网络将振荡，如果大于1，该解将“失效”。因此，同步的必要（但不是充分）条件是 $\varepsilon_k < 1.0$。

在图9-7a中环形网络的基尔霍夫特征值 $\varepsilon_k = -1.0$，因此环形是有界的，但是振荡的。图9-7b中小世界网络的基尔霍夫特征值 $\varepsilon_k = -1.0$，但该网络同步！图9-7中四个网络的基尔霍夫特征值都为1.0，但两个同步而另外两个不同步！为什么？

假设我们沿着由图9-7c所示的网络形成的两个回路中的信号，以右上角的阴影节点开始。网络链路用距离阴影节点的跳数作为标签，沿着两个回路中的路径，最后回到阴影节点。让我们以阴影节点的值定义沿着每个回路每跳的每个节点上的信号强度。起点 s_0 是阴影节点在时间 $t = 0$ 时的状态。沿着两个回路，每一跳后的每个后继节点信号强度是 $s_0/2, s_0/2, s_0/2 + s_0/2 = s_0$，$s_0$，在阴影节点返回了 s_0。经过2跳后，该信号增强了（从 $s_0/2$ 变为 s_0），因为基尔霍夫定律是所有输入的总和。因此，信号以相同强度 s_0 返回到阴影节点。返回到阴影节点的信号强度没有降低，因此网络不同步。

现在考虑一个与混沌网络几乎相同的版本，如图9-7d中所示。这个网络同步！执行相同的实验——从两个阴影节点经过两个回路再回到阴影节点跟踪一个信号。沿着两个回路的路径经过1跳，在直接后继节点上的信号强度是 $s_0/2$，第二跳后的信号仍然是 $s_0/2$，但是注意在第三跳会发生什么。在入度等于2的节点上出现两条路径，但在其他信号之前，信号强度 $s_0/2$ 到达节点，所以从该节点的输出是 $s_0/2$ 而不是 s_0。如此一来，阴影节点接收 $s_0/2$ 作为其下一个状态。阴影节点接收一个减弱的、抑制的输入。减弱的信号强度在其每次沿着回路向前时都会进一步减弱，并返回到阴影节点。这个网络同步，因为它的初始状态是抑制的。我们说这两个回路是*相互干扰*，而不是*彼此增强*的。

我们声称含有增强回路的基尔霍夫网络永远不会同步，而包含干扰回路的网络可能同步，具体取决于干扰的模式。我们进一步采取这种想法，如果2个或更多循环长度（跳数）互质，则循环互相干扰，即包含了两个循环的长度，除了1之外没有其他公因数。

一般来讲，我们可以通过重复增加出度矩阵 $\boldsymbol{B}$，建模一个任意网络回路的信号强度。回顾 $\boldsymbol{B}^k$ 是被 k 跳分开的节点的连接矩阵。强连通网络中，任意两个节点之间的最大跳数是 n。因此，在不超过 n 跳后，信号的强度是 $\boldsymbol{B}^n S(0)$。

如果网络包含循环长度为 k，那么至少有一个对角线 $\boldsymbol{B}^k$ 非零，代表循环强度。此外，k 为循环长度，所以我们只检查 $\boldsymbol{B}$ 中长度为1的循环，$\boldsymbol{B}^2$ 中长度为2的，$\boldsymbol{B}^3$ 中长度为3的，依此类推，直至达到 n 为止。然后，我们对所有的循环对比较，以确定它们是否互质。如果至少有一对是互质的，则网络同步。

例如，图9-7b基尔霍夫网络包含两个循环——一个长度为3，一个长度为5。（3，5）互质，所以这个网络同步。相反，图9-7c不同步，因为它的循环长度（5，5）平分彼此。图9-7d网络同步，因为（4，5）是相对素数对。

为了保证信号强度减弱，当一个信号完成所有的反馈循环时，我们还必须加上条件：至少两个循环的长度是相对素数。这可以确保它们彼此干扰，从而抑制了信号强度。抑制导致网络在零处查找奇异吸引子。

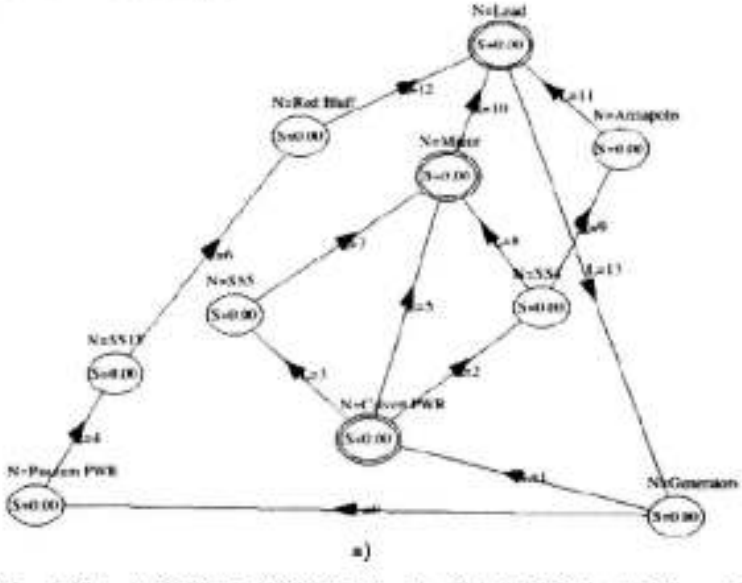
设 $\boldsymbol{B}^k$ 为基尔霍夫出度矩阵，代表 k 跳长度的路径。设 $b_{i,j}(k) = 0$ 是 $\boldsymbol{B}^k$ 的对角线。如果 $b_{i,j}(k) = 0$，长度为 k 的路径存在。因此，基尔霍夫网络同步如果符合以下两项条件：（1）$\varepsilon_k < 1.0$；（2）存在着长度为（k_1,k_2）的循环，使得 k_1 和 k_2 互质。

程序 Network. jar 包含一个使用欧几里得的最大公因子算法的简单方法来决定是否两个数互质。为了方便起见，这里包含一个递归方法 GCF（）。假设 $x \geqslant y$，如果（x,y）互质，则 GCF 返回 1，否则返回 0。例如，给定长度循环（5，3），GCF 返回 1；长度循环（6，3），GCF 返回 0：

```
private int GCF(int x, int y){
     int quotient = x/y;
     int remainder = x - y*quotient;
     if(remainder < 1) remainder = GCF(y, remainder);
     return remainder;
   }
```

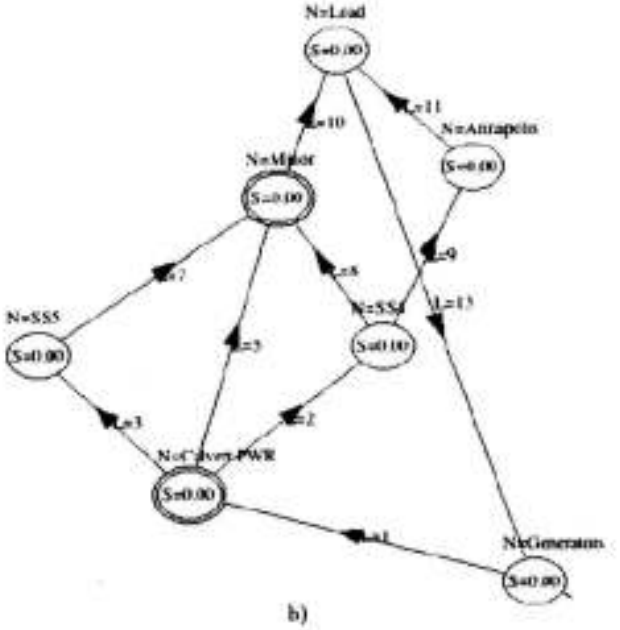
9.4 Pointville 电网

现在，我们将应用本章学习过的同步理论解决很实际的问题——电网的稳定性（或其不稳定性）。这个问题对于所有依赖于电网输配电的工业化国家具有十分重要的意义，如 2003 年 8 月美国东部电网故障，使得 5000 万居民生活陷于黑暗之中。令人惊讶的是，这场大规模的停电，起初只是源于俄亥俄州电网上的一个微不足道的故障。来自俄亥俄州的故障通过加拿大和美国东部传播，影响到远在纽约的人们。

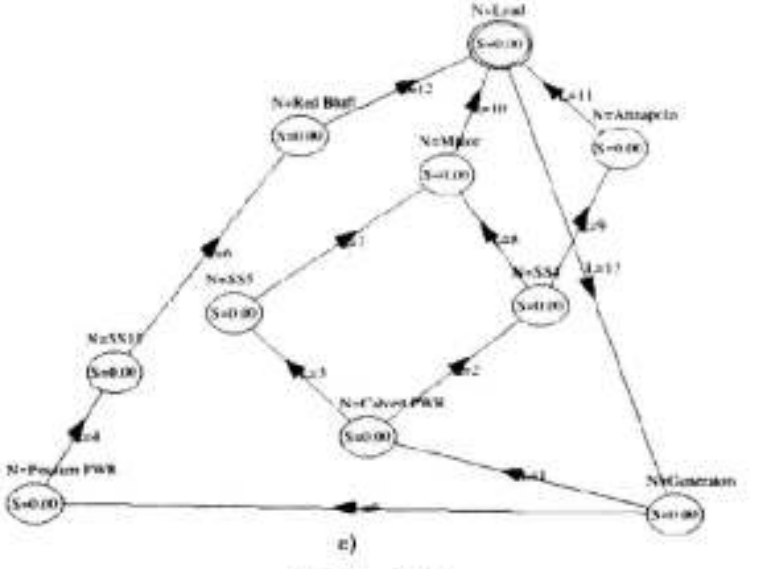


a)

图 9-8 Pointville 电网：a）完全运行的稳定电网；b）发电站故障后（即 Possum 电力发电故障发生后）的部分运行网络；c）单链路故障后，完全不运行的电网



b)



c)

图9-8 （续）

Pointville，Flatland要小得多，但它仍然依赖于图9-8a中所示的电网。像所有电网一样，系统运行人员有着近乎不可能的任务，要求根据随着时间而变的需求维护基尔霍夫回路。每天上午，消费者会消耗大量的电力，但是这种消耗到傍晚当消费者下班回家时就逐渐减少。此外，这种需求随着天气的变化而改变，因为空调会消耗大量的能量，在寒冷的天气比在炎热的天气消耗低。因此，运行人员必须维护使该地区控制误差（ACE）保持在0左右，即网络上每个节点上的流入总和必须等于流出总和。ACE是电力运行人员用来描述基尔霍夫第一定律的术语。

假如ACE <0会发生什么？这种情况意味着没有发出足够的电力来满足需要。运行人员要么必须从备用发电机处购买更多的电，要么让现有的发电机发出更多的电。同样的，假如ACE > 0，运行人员必须抛负荷。消费者的需求会经常发生变化，所以维持ACE = 0是一项困难的平衡行为，需要计算机和人工响应运行人员来完成。

与顾客的需求相比，意外事故更加不可预测。假如一台发电机忽然离线，ACE 会降至 0 以下，运行人员必须通过网络重定向电力来迁就突发的变化。同样，假如 ACE 激增，运行人员必须迅速减少发电或将电力沿着电力线路重定向到其他城市。重新定向具有风险性，如同 2003 年在图 9-8b 中所示的 Possum PWR 发电机故障后会发生剧烈变化一样。从分电厂 SS13 的发电机组和相邻的 Red Bluff 的消费者（一个具有很有竞争力足球队的城市）去掉故障的发电机。

经过 Possum 故障后，Pointville 电网重新稳定并继续运行——而不为 Red Bluff 提供电力——经过一个短期的不稳定，因为其剩余循环的长度互质。删除 Possum、SS13 和 Red Bluff 后留下如图 9-8b 所示的网络，它有两个循环长度，一个为 5，另一个为 4。因为（5，4）互质，在接近 100 个时间步后，部分受损的电网稳定下来。

结果与 Calvert PWR 和名为 Minor 的郊区之间的线路删除有很大的不同（参见图 9-8c）。如果这种电力线路故障，网络就不能同步，因为所有剩余循环的长度为 5，（5，5）不是互质数。Pointville 的电力网完全崩溃，从而造成所有消费者停电！

通过在两个节点之间增加一条链路，Pointville 电网会更有弹性——但是具体在哪些节点之间添加？例如，SS5 和 Red Bluff 之间再建一条链路以防 Possum 故障，但并不能保护它应对如图 9-8b 所示的链路故障。如果 Possum 或 Calvert 和 Minor 之间的链路产生故障，可在 Calvert PWR 和 Red Bluff 之间构建更长的电网来防止灾难性故障，但其花费会更大。这作为一个练习留给读者证明。

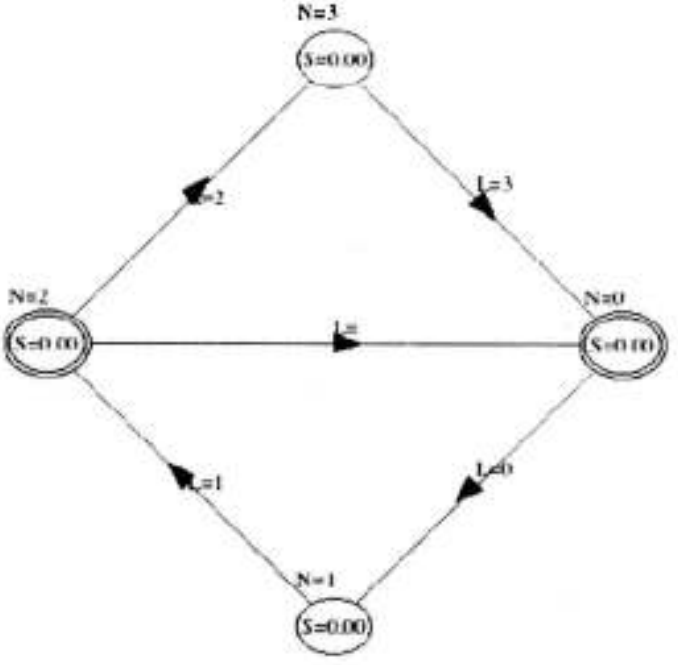


图 9-9 二分环形（小世界）网络；这种网络对于鸣叫、Atay 和基尔霍夫网络算法同步吗？参见课文中的练习 9.9

一般来说，网络的稳定性会随着密度的增加而增加，因为相对互质长度的循环很可能增长。因此，随机网络会随着其密度增加成为比基尔霍夫网络更好的网络，但小世界网络可能无法用相同的方法提高它的稳定性。这个问题留给读者思考研究。

练习

9.1 使用 Z - 变换，证明三角回路是一种稳定的 Atay 网络。

9.2 证明当将一个三角回路添加到任意 Atay 环形网络的任意一个节点上时，它就会同步。

9.3 证明如果至少一个节点状态固定即保持不变时，Atay 网络同步。

9.4 证明当将一个三角回路添加到它的根节点上时，Atay 二叉树网络同步。

9.5 如果固定根节点，Atay 二叉树网络同步吗？

9.6 证明一般的封闭系统基尔霍夫网络，当将一个节点固定时，是不能够同步的。

9.7 对于 $n = 4$ 的基尔霍夫环形网络，出度连接矩阵是什么？

9.8 为什么基尔霍夫环形网络不能同步，而与大小 n 无关？

9.9 图 9-9 中的二分环形网络同步吗？考虑本章中研究过的每一种算法：鸣叫、Atay 和基尔霍夫网络。

9.10 为什么在图 9-8 所示的 Calvert PWR 和 Red Bluff 之间添加一条链路就会提高 Pointville 电网的稳定性？

第10章
Network Science: Theory and Applications

影响网络

影响网络是一个（有向或无向的）网络，$G = \{N, L, f\}$，其中节点为 N，有向加权链路为 L，以及定义有向网络拓扑的映射函数为 $f: N \times N$。影响网络是社会网络的优秀模型，社会网络中节点也称为个体。节点和链路有一个值属性，定义了（链路）影响和与命题相关的（节点）态度或立场（position）。例如，死刑问题是一种命题，每一个节点的态度要么赞成要么反对，链路 $e: v \to u$ 定义了个体 v 对个体 u 态度的影响度 ϕ。典型地，一条（有向）链路 e 的权重等于它的影响度，ϕ = 影响，它是0（无影响）到1（100%影响）之间的一个小数。在这里设计的理论中，影响只会是正的，$0 \leqslant \phi \leqslant 1$，范围为从“无控制”到“完全控制”。

影响网络可以模拟某种社会、经济、政治策略及政策的结果。例如，假设一个影响网路可用于为国家经济建模。一个节点代表税收，其余节点分别代表消费者、就业以及政府开支。链路（影响）代表了一般税收增加或减少对消费者消费、政府花费等的影响效应。这个网络中的每一个节点通过加权链路影响邻接节点。例如，减税可能会增加就业率，反过来又可以增加消费支出，从而又增加了税收，这反过来又可能导致进一步的减税。给定这样的影响网络、影响度 ϕ（链路值）和每个个体的初始态度，我们会问：“增加消费者消费税会导致什么结果?”答案来自于对影响网络稳定输出的研究。例如，节点状态会到达稳定值，还是它们会发生振荡？这种网络的最终状态能够追踪到一个个体对另外一个个体的影响，以及网络的拓扑。

小组决策和竞争者之间的谈判是当今全球经济的主要关注点。假定能对影响精确测量，是否能够预先确定组成员可以达成一致的意见？我们证明了，在一些相当严格限制的条件下，小组达成一致意见不可避免地会从影响网络中涌现。达成一致意见相当于同步，以及同步网络的奇异吸引子值等同于小组的决定。这在实际谈判中及标示网络中最强大的个体时很有意义。

假定小组可以达成一致意见，影响网络中最有影响的个体就是最终控制输出或小组达成一致意见的人。换句话来讲，有用性（power）被定义成对网络具有最大影响的个体。在某种条件下，我们可以猜测为什么一个个体比另一个个体更有用，并确定有用性从哪里来。我们可以证明对其他节点的高度影响，加上来自其他节点的低度影响，等同于 Atay 类型的影响网络中的有用性。

影响网络理论具有很多非常重要的应用，在介绍过影响网络理论后我们会仔细讨论，它是建立在前面章节中研究过的稳定性理论基础之上的。本章中我们会学习两个基本的影响网络，并学习以下内容：

1. 影响网络（简称 I-nets）是包含了称为个体的节点和称为影响的链路的网络。它们建模复杂系统中的社会网络、n - 方谈判和相关性，例如一个国家的经济或生产流程中相互关联的步骤。我们设计开发了达成共识和冲突的理论，并将它应用于确定最有影响的个体问题中，即冲突度最高的个体和网络中具有最大有用性的个体。

2. 研究了 I-nets 的两种主要类型：无向网络，如由 Emanuel Rosen（Rosen，2000）开发的用于研究营销的 buzz 网络；有向网络，如由人类群体中谈判各方形成的社会网络。我们应用前面章节中学习过的同步技术来确定这些 I - nets 是否同步，并且设计开发了关于冲突、达成共识和

有用性的一个新理论，以便解释社会网络中的行为。

3. 一个“buzz 网络”就是一个 Atay 网络，在这里个体的态度由节点的状态表示，其值位于[-1，1]之间，表示正的或负的个体态度。buzz 网络通常是同步的，但是它的奇异吸引子依赖于网络中的度序列。如果 buzz 网络中的节点达到了相同的奇异吸引子值，我们就说网络达成共识。如果奇异吸引子总是为0，就说 I-net 陷入了僵局。典型地，I-nets 不可能达到一个非僵局的状态，因为在个体的群体中至少有一个冲突。

4. buzz 网络达成共识完全由网络的 hub 节点的态度所决定。因此，hub 节点在 buzz 网络中有最大的影响。这是由于 buzz 网络中每个个体的状态是由邻接个体的平均值所决定的。与直觉相反，当最终网络达成共识时，hub 度要比节点的紧度更为重要。

5. 更一般地讲，I-nets 是一个有向的 Atay 网络，在有向链路上带有称为影响的综合权重 ϕ。个体取值于[-1，1]之间的状态，表示反对者态度和赞同者态度，以及两者之间的渐变。如果影响 $|\phi|$ 在[0，1]之间，拉普拉斯矩阵的谱隙小于0（$\sigma<0$），并且在个体之间没有冲突，I-nets 就达成共识（同步）。

6. 我们定义了网络的一个新特性：影响度矩阵 Q，它是从网络的拉普拉斯算子 L 和误差边界 ε 获得的，$Q=[I+L]^{t^*}$，t^* 由以下不等式来定义：$\mathrm{RMS}|[I+L]^{t^*}|<\varepsilon$。函数 RMS[$X$] 根据 X 的每一列元素与每一列的对角元素之差的均方根误差获得的。个体 v_i 的影响度等于 Q 的对角线元素 $q_{i,j}$：$q_{i,j}=\mathrm{degree\_influence}(v_i)$。

7. 我们定义网络的一种新特性：冲突度矢量 C，它等于系统状态矩阵 $[I+L]$ 的对角线，这里 I 是恒等矩阵，L 是拉普拉斯算子。个体 v_i 的冲突度等于 c_i。

仅当 $\mathrm{degree\_conflict}(v_i)=c_i<0$ 时，一个个体才和它邻近的个体相冲突。如果它至少包含一个冲突个体，一个 I-nets 就不能得到一个非0的共识。这种 I-nets 就是僵局。

8. 我们证明了具有非负冲突度 C 的 I-nets 达成一个由影响度矩阵 Q 决定的非0的共识。当 I-nets 中的个体态度不同于其初始态度时，最终达成共识由影响度总和近似确定，该影响度总和通过所有个体的初始态度得出。因此，I-nets 最终的奇异吸引子的值近似等于同意的个体影响度的总和减去不同意的影响度的总和。节点紧度对最终达成共识几乎没有什么影响。

9. 我们在 I-nets 上开发设计了一个有用性的理论，并证明最有用的个体的出链路通常比入链路更多。尽管对于该规则有一些很重要的例外，个体节点通过增加出链路的数量或权重增加了对整个网络的影响。除此之外，减少入链路的权重或数量也可以增加有用性。

10. 我们演示了 I-nets 在社会网络分析中的应用，并演示了如何确定最有用的个体、冲突个体（如果存在的话）以及 I-nets 达成共识的条件。

本章使用了 Nerwork. jar 程序的 buzz 网络部分以及 Influence. jar 程序中的影响、冲突以及紧度分析函数，来产生并分析这里描述的网络。

10.1 对 buzz 的剖析

Emanuel Rosen 在他 2000 年出版的有关新产品营销信息传播的书中，解释了产品市场营销如何使对产品的描述就像网络传染一样传播（Rosen，2000）。利用口碑相传的接触传染要比迟钝的或普通的信息传播快很多，所有的口碑广告通过社会网络从一个人跳到另外一个人。Rosen 将此称为“buzz”，并规定了目标瞄准社会网络中能获得最大 buzz 的那些个体的规则。例如他声称，小世界社会网络的传播速度要比随机网络更快，并描述了各种加快 buzz 传播的技术。

在本节中，我们建立一个 buzz 的模型，通过社会接触进行夸张的广告传播。在广告界中消费者面临着一个建议（命题）——他们是否喜欢或不喜欢某种产品。每个消费者（个体）有一个初始态度——赞成或反对，随着时间的推移，因为有来自朋友和邻居的影响，他们可能会改变

立场。我们将这种过程建模成一个社会网络，其中节点代表消费者（个体），链路代表说服或者影响。我们证明在个体态度上的改变取决于个体的度和固执程度。

我们挑战以下观点：小世界网络与其他网络相比是更好的影响传播者，并且发现 buzz 的剖析大多是针对社会网络的度序列，而不是聚类系数、路径长度或最大介数/紧度。在这里建议的 buzz 模型将产品的正面和负面意见相结合，从而在两种产品之间模拟竞争。此外，我们介绍了固执程度的概念，即拒绝改变个人的观点或信仰。

Rosen 称负面态度比正面态度更强大。一个“坏的影响”可能压制“好的影响”高达 200%。因此，尽快阻止“坏的影响”是很重要的，并用“好的影响”取代它。忽略负面态度的有用性超过正面态度的，假设产品 A 的负面看法是在支持产品 B，而相反，对产品 B 持负面看法是在支持产品 A。因此，我们用负数表示负面态度 $[-1, 0)$，以及正数 $(0, +1]$ 表示正面态度。零表示中立的态度。我们想要知道社会网络中哪种态度强大，正的或者是负的。实际上，如果一个态度可以传播给所有的个体，并稳定在奇异吸引子上，我们就说网络已达成共识。如果没有，我们说网络陷入僵局。换言之，达到网络同步相当于小组达成共识。

我们知道人们被说服的程度不同。有些人容易动摇，而其他人则不容易。假设我们用 $0 \leqslant \phi \leqslant 1$ 表示作为促销活动对象的 129 个 Flatland Pointville 市民的固执因子。当固执因子为 $\phi = 0$ 时（缺乏主见），Flatland 人很容易被说服改变主意。当 $\phi = 1$（固执），Flatland 人不会改变其想法。通常情况下，我们发现 Pointville 本地人是处在两者之间。

Flatland 人趋向于在社团中群体讨论他们的工作、家庭以及消费产品。这似乎是一个很好的消费品公司测试新的营销理念、原型产品以及请求有价值的营销反馈的地方。因此，市场研究人员理解 Pointville 的影响网络的动态特性是很重要的。影响网络的分析将会回答这个问题：“对象是谁，以及怎样抵御负面的评价?”

10.1.1 buzz 网络

令 $0 \leqslant \phi \leqslant 1$ 为所有 Pointville 市民平均固执程度的度量，$(1 - \phi)$ 为附加到普通市民的平均可塑性的权值。此外，令 $S(t)$ 为代表一个个体态度的状态向量：如果态度是负面的，$S(t) = -1$；如果是中立的，$S(t) = 0$；如果是正面的，$S(t) = +1$。一个个体关于产品的态度与思想、政治信仰等有关，通常在这两个极端之间：

$$-1 \leqslant S(t) \leqslant 1$$

$$S(t) = [s_1, s_2, s_3, \cdots, s_n]^{\mathrm{T}} = G \text{ 的状态矢量}; S_i(t) = S(t) \text{ 的 } j \text{ 行}$$

假设 buzz 传播是通过在每个时间段适当修改 Atay 方程模型来进行合适的模拟，从初始条件 $S(0)$ 开始：

buzz 状态方程

$$S(t+1) = \phi S(t) + (1 - \phi) \sum_{j \sim i} \frac{S_j(t)}{\text{degree}(i)}$$

其中 $\sum_{j \sim i}$ 是节点 i 的邻居总和，特别是，$j \sim i$ 和 $s_i = [-1, 1]$，$i = 1, 2, \cdots, n$。

最初，社会网络中所有个体的状态都为 0，除了两个种子节点——一个具有正面态度（+1），另一个具有负面态度（-1）。+1 标志着一个个体是该产品的鼓吹者，而 -1 标志着一个个体是诋毁者或采取相反态度者。中间值如 0.5 或 -0.5，意味着一个未决定的状态，但是具有正的或负的偏向。

每个种子个体向相邻的邻居节点传播其产品宣传，而相邻的节点传播到他们的邻居，如此等等，就像一个传染病的扩散。每个个体的状态取决于个人信念 ϕ 的强度以及相邻节点的态度。

注意聚合的邻居态度是相邻邻居态度上的平均[⊖]。

我们通过标记总和就是 $S(t)$ 与 buzz 网络的连接矩阵的内积来简化状态方程式。然而，我们希望每个节点都能平均，所以我们将度矩阵 $\boldsymbol{B}$ 定义为邻接矩阵除以每个节点的度：

$$\boldsymbol{B} = \begin{pmatrix} 0 & \frac{1}{\text{degree}(1)} & \cdots & \frac{1}{\text{degree}(1)} \\ \frac{1}{\text{degree}(2)} & 0 & \cdots & \frac{1}{\text{degree}(2)} \\ \cdots & & 0 & \\ \frac{1}{\text{degree}(n)} & \cdots & \frac{1}{\text{degree}(n)} & 0 \end{pmatrix}$$

然后，$S(t+1) = \phi S(t) + (1-\phi)\boldsymbol{B}S(t) = [\phi \boldsymbol{I} + (1-\phi)\boldsymbol{B}]S(t)$，给定 $S(0)$。

这种状态方程将在稍后进行探讨，但现在注意 $\boldsymbol{B}$ 明确为正的，这意味着它有正的特征值。因此，这种 buzz 网络总是同步的，所有节点收敛到同一值。buzz 网络是稳定的，但它的奇异吸引子值可能为零也可能不为零。根据正和负的种子的位置，社会网络的最终状态可能达到零或区间 [-1, 1] 之间的值。具体来讲，社会网络的度序列完全决定 Pointville 消费者能否达成某种共识。这个重要的细节是理解 buzz 网络工作原理的关键。

10.1.2 buzz 网络仿真器

Network.jar 程序实现了以上描述的如病毒一样的 buzz 过程。它可以传播正的或负的影响。此外，用户既可以任意选择一个起始节点，也可以让程序选择一个随机的或 hub 节点。无论用户选择正或负的态度，程序会选择相反的。因此，网络将达成一个共识，它取决于节点的初始状态和网络的拓扑结构。

方法 NW_doBuzz(int done)非常类似于早期研究的 NW_doKirchhoff()。第 1 趟更新每个节点的当前状态到下一个状态。如果当前和下一状态之间的相对差别很小，变量 stable_time 会增加，这表明涌现的进程尚未稳定下来。另一方面，如果 stable_time 仍然是零，这表明几乎所有的节点都是稳定的，该方法的增加会返回变量 done。调用的方法然后会在调用 NW_doBuzz 之前通过将 done 设置为 0 来确定收敛到网络的稳定性，并检查返回值。返回值 1 表示网络已经稳定下来了。

第 2 趟更复杂，因为它平均了相邻节点的值——为网络的每个节点——然后以平均值来递增每个节点。节点以随机顺序处理以删除仿真偏差。Network.jar 程序借用以前用于模拟传染的变量 InfectionRate，存储固执程度值 ϕ。这个参数不能超过 1，既然状态变量也不能超过 1，每个节点的 next_state 的值根据需要处于 -1 到 +1 之间。此外，还由于 buzz 网络是封闭系统，每个节点的度必须大于零。因此，我们不必担心被 0 除的例外情况。

最后，方法 NW_doBuzz()修剪 next_state 的值来加速收敛到一个吸引子。随着网络达到吸引子，状态的改变单调地逐渐减小（将在下一节介绍）。因此，修剪方法不能改变网络的必然状态。然而，我们发现奇异吸引子不等于零，它们可以等于区间 [-1, +1] 之间的任何值：

```
public int NW_doBuzz(int done){
    int stable_time = 0;
    int tail_node, head_node;              //Tail and Head of Links
    for(int i = 0; i < nNodes; i++) {   //Pass 1: Update to next_state
       double ns = node[i].next_state;  //Future state
       if(Math.abs((node[i].next_state - ns)/ns) > .00001)
            stable_time++;                //Not stable yet
```

⊖ 邻接节点的当前值总和除以度 degree(i)，这与所有邻居节点的平均相同。

```
            node[i].value = ns;          //Update to future state
        }
        if(stable_time == 0) done++;       //No state changes
        //Randomize processing order
        Shuffler random_list = new Shuffler(nNodes);
        for(int i = 0; i < nNodes; i++) random_list.card[i] = i;
        random_list.SH_Shuffle(nNodes);
        for(int n = 0; n < nNodes; n++){   //Pass 2: Next State
         int i = random_list.card[n];      //Scrambled order
         double neighbor = 0.0;
         node[i].next_state = InfectionRate*node[i].value/100.0;//RHS
         for(int j = 0; j < nLinks; j++){ //Across all links to this node
            tail_node = Link[j].tail;
            head_node = Link[j].head;
            if(tail_node == i) {           //All neighbors
                neighbor += node[head_node].value;
            }
            else if(head_node == i) {
                neighbor += node[tail_node].value;
            }
      }
      node[i].next_state += (1.0-InfectionRate/100.0)*neighbor/
                            node[i].degree;
      //Clip next_state
      if(Math.abs(node[i].next_state) <.00001) node[i].next_state = 0;
    }
    return done;
  }//NW_doBuzz
```

10.1.3 buzz 网络的稳定性

在 Flatland 的营销世界里正面的和负面的产品宣传像传染病一样迅速传播，直到达到 Pointville 的 buzz 网络中的每个个体。消息是好还是坏并不重要，但是它是否到达每个人，影响了大多数的人口，这对广告者会有影响。假设 Pointville buzz 网络是一个封闭的系统，它是否同步？如果所有节点达到相同的最终状态，我们说网络已达成共识。对于 Pointville 网络达成共识的值是什么？这些问题对于花数百万美元来推广其产品的营销主管是很重要的。

我们证明了 buzz 网络唯一要紧的属性是度序列。此外，我们证明了与达成共识有关的度序列属性只是识别 hub 或最高度的节点。与市场营销的某些思想相反，社会网络具体是随机的、小世界的或无标度的并不重要。此外，Rosen 声称最为重要的紧度属性对于说服力的影响不大，因为节点的介数/紧度与快速扩展口碑的能力无关。相反，hub 节点是最重要的个体。每个网络至少有一个 hub，这个 hub 正是市场营销人员应该控制的，以便取得最好的结果。我们证明了 hub 决定着所有其他个体的最终态度。因此，hub 是宣传接种的最佳位置。

假设 Pointville 网络的 129 个节点的状态最初时设置为零，指示有关某一产品的中立意见。现在假设随机选择一个节点为正面宣传产品，称此为促销者，随机挑选第二个节点为负面宣传产品，称此为反对者。在 Network. jar 程序中，我们设置促销者的初始状态为 +1，反对者的为 -1。经过充足的时间后，Pointville 中个体节点的最终态度是什么？

我们对整个网络重复应用状态方程——根据状态方程更新每一个节点。首先，网络是否同步，如果是这样，所有节点的最终值是什么？更加切题的是：个体节点之间是否相互保持一致，

如果是的话，他们成为促销者还是反对者？作为市场营销人员，我们想对一个个体接种正面态度（+1），并将它传播到所有其他个体。然而，如我们显示的，这取决于最初的促销者或反对者节点的度的相对大小。种子节点的度越高就会越强大，较高度的节点要胜过较低度的节点。

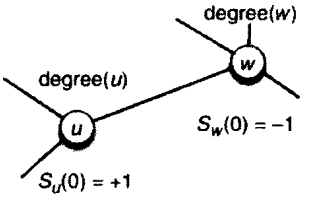


图 10-1 描述了 buzz 网络的状态方程：更高度的节点支配较低度的节点

图 10-1 说明了当促销者和反对者由共同的链路连接起来时会发生什么，以及对其他节点这样做后有什么影响。假设其他节点的初始状态是零，我们计算节点 u 和 w 的下一个状态，使用状态方程：

$$s_u(1) = \phi s_u(0) + (1-\phi)\frac{s_w(u)}{\text{degree}(w)}; \text{给定 } s_u(0), s_w(0)$$

$$s_w(1) = \phi s_w(0) + (1-\phi)\frac{s_u(0)}{\text{degree}(u)}$$

现在我们通过将两个方程加起来除以 2 来平均两个状态，让合并后的状态方程的平均值由 $s_{avg}(t)$ 指定：

$$s_{avg}(1) = \phi s_{avg}(0) + (1-\phi)\frac{\frac{s_w(0)}{\text{degree}(w)} + \frac{s_u(0)}{\text{degree}(u)}}{2}$$

或者，更一般地：

$$s_{avg}(1) = \phi s_{avg}(t) + (1-\phi)\frac{\frac{s_w(t)}{\text{degree}(w)} + \frac{s_u(t)}{\text{degree}(u)}}{2}$$

对于这种差分方程的特征解，当 $\phi < 1.0$ 时，具有一个由 ϕ^T 给出的呈指数下降的组件部分。特殊解有单调减少、增加或常数项，具体根据 $F(t)$ 的符号来判断：

$$F(t) = s_u(t)\text{degree}(u) + s_u(t)\text{degree}(w)$$

$$\text{初始时 } F(0) = s_u(0)\text{degree}(u) + s_u(0)\text{degree}(w)$$

换言之，状态方程的解包含一个逐渐消失的组件和一个趋向 0、1 或 -1 的组件，它取决于初始条件以及 u 和 w 的度。表 10-1 列举所有解的可能组合，忽略了逐渐消失的组件。

表 10-1 buzz 网络可能的最终状态

degree (w) vs. degree (u)	$s_w(0) = +1, s_u(0) = -1$	$s_w(0) = -1, s_u(0) = +1$
w > u	+	-
w < u	-	+
w = u	0	0

在图 10-1 中节点 u 和 w 的最终状态由每一个节点度的相对大小来确定。假设一个节点是促销者（+1），另一个是反对者（-1），种子节点的相对度确定了网络中所有节点的最终状态！例如，如果 degree (w) > degree (u)，并且 $s_w(0) = +1$，$s_u(0) = -1$，buzz 网络将倾向于正的吸引子（+）。但是，如果 degree (w) = degree (u)，网络将随着状态趋于零而同步，而与网络的度序列无关。如果它们具有相同的度，最初的促销者和反对者的态度会相抵消。

这一结果通过模拟来验证。例如，在环形或2－规则网络里的所有节点具有相同的度。这意味着奇异吸引子应该趋向于0，因为所有的节点都是一样的。2－规则网络的奇异吸引子为0，而达成共识的环形网络是$1/n$。因此，环形buzz网络将在n无限增加的极限下同步到0。

如果促销者和反对者节点的度都是相等的，那么完全、超环形、超立方、随机、小世界以及无标度网络趋向0（中立达成共识）。仿真研究也可以验证当接种正的或负的初始态度时，hub节点在随机、小世界以及无标度网络中占据主导。如果hub节点接种成（－1），网络就不可能达成共识；如果接种成（＋1），那么网络就能达成共识。因此，buzz网络达成共识是由主导（hub）节点影响的。与直觉相反，更高度的hub将赢得具有较低度的最大紧度的节点。中间节点不如高度连通的节点有用性高，因为更高连通的节点有助于小世界网络效应，它们的态度要比其他节点更快地传播到其他节点。

市场营销的结果是显而易见的，整个网络受到单一节点的影响，而单一节点的连接越多，影响就越大。最好的营销策略是找到hub并使用正的或负的方法来控制它。随着hub的正负改变，整个网络也随之改变。

在将此规则应用到*开放系统*时会有一些例外。例如，一个线性网络是开放的（因为它的终端节点）。终端节点的度为1，而中间节点的度为2。因此，中间节点应该对整个网络有更大的影响。但是，事实证明，中间节点在线性网络达成共识时只有很小的影响，即使它们比最终节点具有更高的度。为什么？对这一现象的解释留给读者作为练习。

10.2 社会网络的有用性

buzz网络是无向的，一个个体节点作用在另一个节点上的影响严格地取决于网络的度序列。这种约束限制了buzz网络应用到更为简单的社会网络，以及对带有交互部分的系统建模的网络，例如国家经济和复杂的制造过程。在本节，我们除去这种限制并提出了I-nets（*影响网络*）的一般理论，它结合了有向Atay网络的拓扑结构和称为影响的有向加权链路。I-nets更为通用，因为每条链路都分配了一个不同的权值：

影响网络（I-nets）。$G(t)=\{N(t),L(t),f\}$，其中$N(t)$是一个随着时间变化状态的个体集合，$S(t)$是$[-1,+1]$之间的状态，$L(t)$为具有介于$[0,1]$之间的权重ϕ的影响集合，并且f是一个$N\times N$的静态映射——即G的拓扑图。

I-nets *状态方程*。影响网络行为上与有向Atay网络相似，但具有不同的影响。个体节点状态的变化率是由相邻节点之间的状态差异来决定的：

$$s_i(t+1)=s_i(t)+\sum_j[s_j(t)-s_i(t)]\phi^{\mathrm{T}}i,j;\text{给定 } s_i(0)$$

其中ϕ^{T}是从网络的拓扑结构和分配给链路的影响权值来获得的。

这个I-nets的扩展定义是对下一个状态方程定义的扩展。下一个状态是不同的权值和相邻个体的不同态度的函数。不是平均邻居的值，这种下一个状态函数计算了差异。从逻辑上讲，当相邻节点之间的态度没有区别时，I-nets达成共识。因此，当所有个体节点的分歧缩小到零时I-nets达成共识！

下面我们证明了如果谱隙$\sigma(L)$为负，I-nets指数地达成共识（奇异吸引子）。此外，我们证明了达成共识（奇异吸引子）的值是一个由置换权重矩阵ϕ^{T}和初始化状态矢量$S(0)$确定的固定点。当达成共识且保证它是非零时，我们特别感兴趣估计一个任意的I-nets的奇异吸引子。

该理论应用到社会网络分析中是显而易见的——我们可以预测在一个社会群体的成员之间谈判的可能的结果，以及在一个复杂的系统中一个或多个个体节点的影响。相反，我们可以估算在何种条件下一个复杂的系统将永远达不成一个非零的共识。在这两种情况下，我们还可以决

定哪个个体节点有更多的有用性，哪个或哪些个体节点阻止了过程达成共识。

10.2.1 两方谈判

一个简单的I-nets包含了两个互相谈判的个体，如图10-2所示。两个个体（两方）被显示成节点。从个体节点0:Us到个体节点1:Them的链路标记成33%，也就是说，节点0对节点1的影响为33%。同样，从个体1到个体0的50%的链路表示节点1对节点0的影响量。一个个体节点作用在另一个个体节点上的影响是通过连接链路的权值乘以源个体节点的状态来计算的，并将乘积转发到目的地个体节点。

影响可通过计算个体同意其他人的次数百分比，或通过收集其他历史数据来确定。例如，影响的估计可能会通过投票计算两个个体相互达成共识的次数百分比来确定。另一种策略可能涉及对谈判的观察，以确定多少次数百分比可以让一个个体打败另一个个体。一般来讲，影响权值用于衡量直接连接的个体达成共识的次数。

随着时间的推移，谈判是一个交互妥协的过程。这个过程通过在很多时间步重复迭代下一个状态函数来模拟。在每一步，所有个体节点的状态是根据下一状态函数（不同）来更新的。我们希望，谈判达到一个奇异吸引子或达成共识。如果这样的话，在每次互动后，每个个体节点就会更接近对方的态度。位移量取决于影响链路的值和个体节点之间态度的不同。这正有Atay状态方程所精确建模的。

假设这两个个体在时间$t=0$时正好在直径上是相对的。让个体节点0：Us初始化采取正的态度+1，而1：Them采取负的态度-1。个体节点的初始状态是Us(0)=1，Them(0)=-1。最初，从Us的角度看他们之间的态度的不同是(-1 -1)=(-2)，而从Them的角度看是(+1-(-1))=(+2)。谈判导致分歧或递归地反复减少到0。

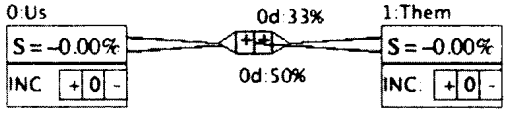


图10-2 I-nets代表两方谈判：节点Us与节点Them，各自的影响为1/3和1/2

两方谈判是一个逐步交互妥协的过程，如果希望网络达成共识，两个个体必须缩小同对方态度的差异。也就是说，每一步后，节点Us必须移动到Them的态度，移动量由50%的影响链路确定，而Them必须移向Us（由其差距的33%确定）。重复应用下一个状态函数并不能保证达到零状态，但它可能导致差距为0。

例如，经过每个时间步后：

$$\mathrm{Us}(1)=\mathrm{Us}(0)+50\%(\mathrm{Them}(0)-\mathrm{US}(0))=1+50\%(-2)=0$$

$$\mathrm{Them}(1)=\mathrm{Them}(0)+33\%(\mathrm{US}(0)-\mathrm{Them}(0))=-1+33\%(2)=-33\%$$

对于每一个时间步重复应用下一个状态函数，直至两方达成共识，或者他们陷入僵局为止。当然，我们想知道的是双方的最后态度，或者更重要的是，我们想知道是否双方曾在中间碰面。通过反复应用状态方程：

$$s_i(t+1)=s_i(t)+\sum_j[s_j(t)-s_i(t)]\phi^{\mathrm{T}}i,j;$$

我们可以确定ϕ^{T}为影响矩阵的转置矩阵

$$\mathrm{Us}(2)=\mathrm{Us}(1)+50\%(\mathrm{Them}(1)-\mathrm{Us}(1))=0+50\%(-0.33)=-0.167$$

$$\mathrm{Them}(2)=\mathrm{Them}(1)+33\%(\mathrm{Us}(1)-\mathrm{Them}(1))=-0.33+33\%(0.33)=-22\%$$

$$\mathrm{Us}(3)=\mathrm{Us}(2)+50\%(\mathrm{Them}(2)-\mathrm{Us}(2))=0.167+50\%(-0.056)=-19\%$$

$$\mathrm{Them}(3)=\mathrm{Them}(2)+33\%(\mathrm{Us}(2)-\mathrm{Them}(2))=-22+33\%(0.056)=-20\%$$

两个个体节点的状态收敛，这意味着它们达成共识——额外应用下一个状态函数使得个体

在相同状态。在这个例子中，分歧降低到零，双方个体达成共识，等于负的20%，这倾向于Them的初始态度。因此，节点1：Them比节点0：Us更有影响。

在一般情况下，共识是由初始态度和影响矩阵确定的，影响矩阵是通过转置加权连接矩阵获得的。在这个简单的例子中，影响矩阵 $\boldsymbol{\phi}^{\mathrm{T}}$ 按如下方式获得：

$$\boldsymbol{\phi} = \begin{pmatrix} 0 & 0.33 \\ 0.50 & 0 \end{pmatrix}$$

$$\boldsymbol{\phi}^{\mathrm{T}} = \begin{pmatrix} 0 & 0.50 \\ 0.33 & 0 \end{pmatrix}$$

10.2.2 I-nets 状态方程

I-nets近似于社会网络内部的相互作用，谈判尝试朝双方都满意的态度努力——达成共识。但是，它总是能达成共识吗？此外，谁在谈判拥有更多的有用性？要回答这些问题，我们正式定义个体的*有用性*为一个个体超过个体状态最终结果的*影响度*。

假设社会网络中的每个个体节点采取一个初始态度，如果反对命题则（-1），而赞同则（+1）。我们将不同个体节点之间的分歧量化成其态度的不同，$(s_j - s_i)$。在个体 v_i 和 v_j 之间的影响度 $\boldsymbol{\phi}(j,i)$，是 [0，1] 之间的分数，代表这两个个体“中途相遇”或在谈判中妥协的能力。在此模型中，个体 v_i 在每次互动后改变其态度的量为 $\boldsymbol{\phi}(j,i)(s_j - s_i)$：

$$\Delta s_i(t) = \boldsymbol{\phi}(j,i)(s_j - s_i)$$

如果个体 v_i 受到多个相邻个体节点的影响，那么将来状态更改就是作用于个体上的所有影响的总和。

$$\Delta s_i(t) = \sum_j [\boldsymbol{\phi}(j,i)(s_j - s_i)]$$

这里 j 是 v_i 的邻居范围。使用 $\phi(j,i)$ 的转置更容易，所以我们将 $\boldsymbol{\phi}^{\mathrm{T}}(i,j)$ 替换到状态方程中：

$$\Delta s_i(t) = \sum_j [\boldsymbol{\phi}^{\mathrm{T}}(i,j)(s_j - s_i)]$$

其中 j 为 v_i 的邻居范围。差异的总和等于总和的差异，所以我们可以通过影响矩阵的*拉普拉斯算子*取代影响矩阵。以矩阵的形式表示，状态方程为 $\Delta S(t) = \boldsymbol{L}S(t)$，其中 $\boldsymbol{L}$ 是 $\boldsymbol{\phi}^{\mathrm{T}}$ 的拉普拉斯算子。

图10-3a显示了包含三个个体节点的简单社会网络I-nets模型——由猎犬（Hound）、狐狸（Fox）、松鼠（Squirrel）组成，影响分别为1/3、1/2、3/4。这种社会网络的关键属性是由初始状态矢量 $S(0)$ 和影响矩阵 $\boldsymbol{\phi}$ 来概括的，我们通过行求和来转置并变换成它的拉普拉斯算子：

$$S(0) = [0,0,0]^{\mathrm{T}}$$

$$\boldsymbol{\phi} = \begin{pmatrix} 0 & 0.33 & 0 \\ 0 & 0 & 0.5 \\ 0.75 & 0 & 0 \end{pmatrix}$$

$$\boldsymbol{\phi}^{\mathrm{T}} = \begin{pmatrix} 0 & 0 & 0.75 \\ 0.33 & 0 & 0 \\ 0 & 0.5 & 0 \end{pmatrix}.$$

通过将行之和插入到对角线元素，将转置影响矩阵变换为拉普拉斯矩阵：

$$\boldsymbol{L} = \begin{pmatrix} -0.75 & 0 & 0.75 \\ 0.33 & -0.33 & 0 \\ 0 & 0.5 & -0.5 \end{pmatrix}$$

如果状态方程 $\Delta S(t) = \boldsymbol{L}S(t)$，I-nets就会同步，会有负的特征值——也就是必须有负的谱隙 $\sigma \boldsymbol{L} < 0$，但这不足以保证I-nets的稳定性。

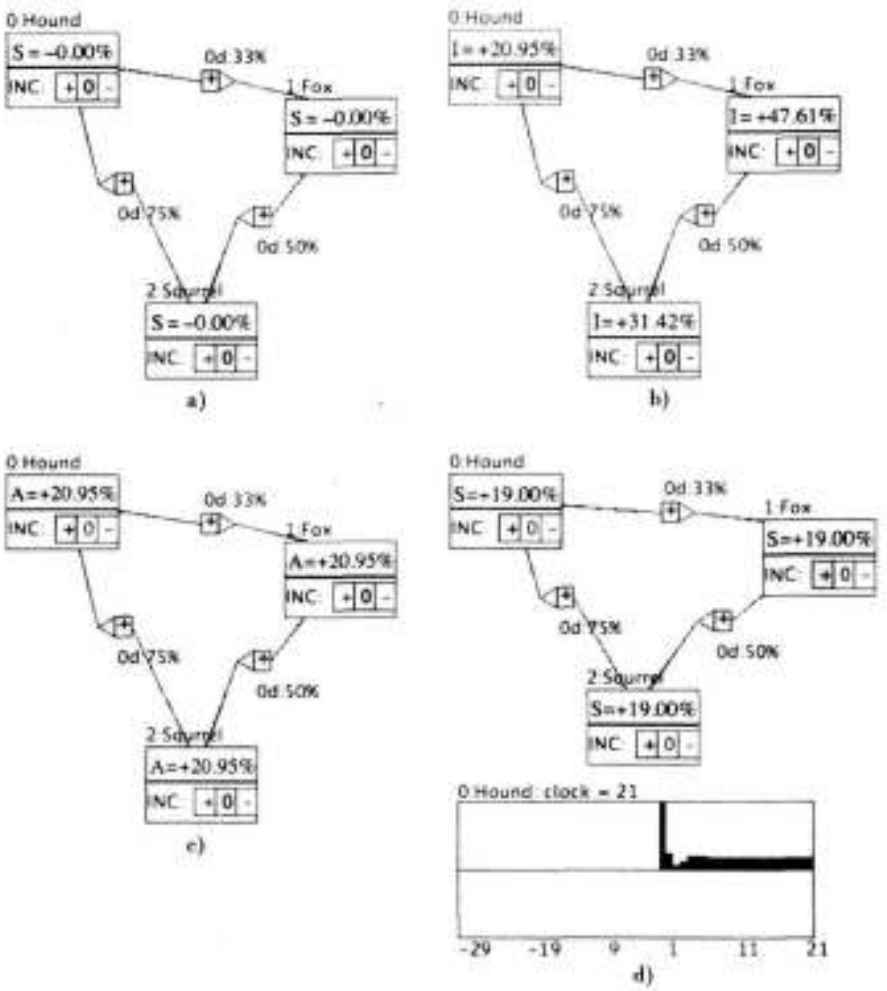


图 10-3 简单社会网络的 I-nets 模型：a) I-nets 显示影响；b) 每个个体节点上显示的影响度；c) $S(0) = [1,0.0]^{\mathrm{T}}$ 的奇异吸引子的值在每个个体节点上显示；d) 评估结果的样例显示

注意状态方程的解，$S(t+1) = S(t) + LS(t) = [I + L]S(t)$，是指数之和 $E|S(t)| = [I + L]S(t)$，其中，E 是时移运算符，$E|S(t)| = S(t+1)$。

简化后，我们就得到：

$$[(E - I) - L]S(t) = 0$$

对于这种运算符方程的特征解就是如下方程的根 r_i 的加权和，这里 $i = 1,2,\cdots,n$：

$$[(r-1)I - L] = 0$$

根集合构成解的基本集合，$S(t) = \sum^{i,t} c_{i,j} r_i^t; t = 1,2,\cdots$。注意这些根与拉普拉斯算子 L 的特征值 ε 有关，由此可得到根如下所示：

$$[\varepsilon I - L] = [(r-1)I - L], \text{所以 } r - 1 = \varepsilon, \text{或 } r = 1 + \varepsilon$$

因此，每个根都要比相应的 L 的特征值更大。也因为解是指数幂的线性组合，从初始值 $S(0)$ 开始，个体节点的状态呈指数增加或减少。一个个体节点的最终状态取决于状态矩阵 $[I + L]$ 的特征值，并且既然 L 是由 I-nets 的拓扑和影响权值来确定，我们可以通过分析 $[I + L]$ 来估算谈判的结果。

10.2.3 I-nets 的稳定性

从第9章中我们知道当网络的所有节点都达到相同的值时就会实现同步。在建模成 I-nets 的社会网络中，同步相当于达成共识。我们也知道，在 Atay 网络中达到同步取决于它的拉普拉斯矩阵的谱隙。因此，如图 10-3 所示的社会网络达成共识的必要（但不是充分的）条件为 $\sigma(\boldsymbol{L}) < 0$，其中 σ 是 $\boldsymbol{L}$ 的最大非平凡特征值。

达成共识的收敛速度像指数增长一样快，因为它的状态方程的解是指数增长的线性组合：也就是根受限于1，因为 $r = \varepsilon + 1$。如果最大的根刚好为1，解包含指数衰减的项，解就不会“失效”，因为包含根等于1的项。因此，I-nets 总是收敛到某一奇异吸引子但并不总为0。

考虑前面的推导和图 10-3c、图 10-3d 中的网络。$\boldsymbol{L}$ 的特征值分别是 $\varepsilon_0 = 0$，$\varepsilon_1 = -0.79 + 0.4\mathrm{i}$，$\varepsilon_2 = -0.79 - 0.4\mathrm{i}$ [⊖]。L 的非平凡特征值都是负的，因为它们是带有虚部的复数，在稳定到奇异吸引子之前同步振荡。在这种情况下谱隙是最大非平凡特征值的实部，$\sigma = \varepsilon_1 = -0.79$。社会网络达成共识，因为 $-0.79 < 0$ [⊜]。或者，如果谱隙大于零，即 $\sigma > 0$，I-nets 将“失效”，而不会达成共识[⊜]。

图 10-3c 显示了初始条件 $S(0) = [1,0,0]^{\mathrm{T}}$ 的理论奇异吸引子。所有个体达到 20.95% 的共识。不同的初始条件 $S(0)$，就会达成不同的共识。小组的理论共识可以很容易从拓扑、影响和初始状态 $S(0)$ 中计算出来。由 Influence.jar 程序来完成，使用由用户设置的初始值。虽然理论共识只是一个近似，但它能足够精确地测量单个个体的有用性。这是在非冲突网络中达成共识的上限。

图 10-3d 中显示了初始条件为 $S(0)$ 的三角或三方 I-nets 的评估结果。实际达成的共识是19%而不是理论值 20.95%。一般来讲，理论估计略高于实际达到的共识。请记住，实际值是根据几项 I-nets 的限制条件达成的。我们在后面会详细讨论这些限制。

图 10-3d 也显示了针对不同时间的个体“猎犬”的状态图。注意初始值如何急剧下降到奇异吸引子。这证实了解是指数地收敛的理论预测。此外，如果 I-nets 状态方程解包含虚根，它的解在收敛时就会振荡。这也证实了上述分析研究的理论。

10.2.4 I-nets 的共识

社会网络分析是对网络中说服力有用性的研究，但在文献中有许多不同的社会有用性的定义。我们选择定义社会有用性为：个体对同步网络的最终值（奇异吸引子）的总的影响。社会有用性可以由随着时间趋于无穷大时的状态方程的渐近解来近似：

$$S(t+1) = S(t) + \boldsymbol{L}S(t); \text{给定 } S(0) = [\boldsymbol{I} + \boldsymbol{L}]S(t)$$

因此

$$S(t) = [\boldsymbol{I} + \boldsymbol{L}]^t S(0); t = 1,2,\cdots$$

设 $\boldsymbol{Q}$ 是有用性矩阵，对应于每个个体节点对 I-nets 网络的最后状态（达成共识）的影响。Q 是系统状态矩阵，提高到某一“大的”有用性。随着时间的无限增大，并且如果 $\sigma < 0$，则 $S(t)$ 收敛到奇异吸引子：

$$s(\infty) = [\boldsymbol{I} + \boldsymbol{L}]^{\infty} S(0)$$

但是我们不可能无限地计算下去，因此我们用 t^* 近似如下：

$$\mathrm{RMS}([\boldsymbol{I} + \boldsymbol{L}]^{t^*}) < \varepsilon$$

⊖ 这里 i 是（-1）的虚平方根，而 ε 是 $\boldsymbol{L}$ 的特征值矢量。

⊜ 我们将在下一节讨论这种结论的例外情况。

⊜ 程序 Influence.jar 修剪状态值，因此状态衰减到零而非无穷大。

$$\boldsymbol{Q} = [\boldsymbol{I} + \boldsymbol{L}]^{t'}$$

这里 ε 一般为0.0001。函数 RMS（）定义成均方根误差，通过从每个其他列元素中减去 $\boldsymbol{Q}$ 的每一列的对角元素，求平方差、求和，然后再取和的平方根得到：

$$\mathrm{RMS}(X) = \mathrm{sqrt}(\sum (x_{i,j} - x_{i,j})^2)$$

例如，对于图10-3b的双方谈判网络，t^* 为14，产生影响度矢量 [0.21，0.48，0.31]：

$$\boldsymbol{Q} = [\boldsymbol{I} + \boldsymbol{L}]^{14} = \begin{pmatrix} 0.21 & 0.48 & 0.31 \\ 0.21 & 0.48 & 0.31 \\ 0.21 & 0.48 & 0.31 \end{pmatrix}$$

图10-3b中的个体根据影响度矢量进行排序如下（最强的优先）：

个体1：狐狸，48%

个体2：松鼠，31%

个体3：猎犬，21%

就社会网络分析而言，这意味着狐狸的态度将超过其他两个中的任何一个。但是如果两个个体参与者投票反对第三个，会发生什么？在这种情况下，多数为胜（参见表10-2）。效果上，像投票一样影响度是累计结果确定 I-nets 的共识。然而，从 $\boldsymbol{Q}$ 得到的理论值只有对实际达成共识的近似，如表10-2所示。

在表10-2中共识是个体节点初始状态的函数。在表的上半部分（无异议），两个节点最初设置为（+1），而第三个设置为0。例如，个体节点“0&1”设置为（+1），它们的影响度添加到一起获得69% =（21% + 48%）。实际的共识值通过使用 Influence. jar 模拟得到。

同样，“0&1 vs. 2”达成共识的近似是通过增加“0&1”的影响并减去反对者的影响“2”获得。因此，对这个初始条件设置达成共识的近似是37% =（21% +48% -31%）。通过模拟得到的实际共识是34%，这要低得多。一般来说，近似比实际达成的共识更高。更复杂的网络将有更复杂的增减。例子留给读者作为练习。

表10-2 图10-3达成的共识

个体	近似（%）	实际（%）
无异议		
0&1 vs. 2	69	66
0&2 vs. 1	52	50
1&2 vs. 0	79	76
有异议		
0&1 vs. 2	37	34
0&2 vs. 1	8	3
1&2 vs. 0	58	55

10.2.5 计算影响的 Java 方法

计算一个个体节点的影响度可直接应用矩阵乘法。在本书中使用的所有矩阵计算工作由 JAMA 提供的可重用的类执行，所以在需要时我们只是从 JAMA 包调用标准矩阵乘法和特征值的方法[⊖]。

首先，我们需要一个 Java 方法用来创建包含 ϕ 中的影响的连接矩阵。然后方法 Diagram_

⊖ 美国数学学会期刊（JAMA）用于线性代数的软件包

doInfluences（）计算每个个体节点的理论影响度，通过重复 $[I+L]$ 乘积来获得 $Q=[I+L]^{t^*}$。有用性的计算是通过重复乘积直到 RMS $[I+L]^{t^*}<\varepsilon$ 得到。

方法 ConnectionMatrix()构造矩阵 ϕ，这是通过将影响权值复制到 Java 矩阵 M 中实现的。该方法比我们在本章需要的目的更加普遍，因为它允许权值是正或负。我们假设所有的权值是正的，所有链路都是单向的。该 Matrix 类由 JAMA 包提供并支持访问方法 M. get()和 M. set()——获得并设置矩阵 M 的元素的操作：

```
//Directed Connection matrix with weights
private Matrix ConnectionMatrix() {
    Matrix M = new Matrix(nActors, nActors, 0.0);
    for(int i = 0; i < nInfluences; i++){
            int sign = 0;
            if(Influences[ i].v > 0) sign = 1;
            else if(Influences[ i].v < 0) sign = -1;
            double d = sign*Influences[ i].weight/100.0;  //Tail-to-head
            M.set(Influences[i].from, Influences[i].to, d);  //Connection
    }
    return M;
}//ConnectionMatrix
```

方法 Diagram_doInfluences()执行所需的矩阵乘法以便计算理论影响度，并将它们插入到 I-nets 网络中。该方法对个体节点有直接的负面影响，在 Actors[I]. Actor_influence 中返回影响度，并返回最大影响值和个体节点的数量。最后，当参数 use_initial_value 为 true（真）时，这个方法还计算了对应于初始条件 $S(0)$ 的奇异吸引子值。这些结果通过数组 Actors［row］. next_ v 中继回到调用程序。程序 Influence. jar 显示影响或奇异吸引子，具体根据用户在控制面板上按哪个按钮而定：

```
//Calculate (I+L)^t*
public MatrixReply Diagram_doInfluences(boolean use_initial_value) {
    MatrixReply reply = new MatrixReply();
    //M = (I+L)
    Matrix M = ConnectionMatrix();
    M = M.transpose();                    //Influences
    for(int row = 0; row < nActors; row++){
        double sum = 0.0;
        for(int col = 0; col < nActors; col++)
            sum += M.get(row, col);
        M.set(row, row, 1.0-sum);     //I+Laplacian
    }
    double rms = 1.0;                     //root-mean-sq error bound
    Matrix I = M;
    int count = 10*nActors;               //Avoid infinite loops
    while(rms > 0.0001 && count > 0){
        I = I.times(M);                   //Powers of (I+L)
        rms = ColumnRMS(I);               //Over all columns
        count--;
    }
    //Store in I-network
    reply.max_value = I.get(0,0);
    for(int i = 0; i < nActors; i++){
```

```
            double influence = I.get(i,i);
            Actors[i].Actor_influence = influence;
            if(reply.max_value < influence){
                    reply.max_value = influence;
                    reply.row = i;
            }
    }
    reply.number = 10*nActors-count+1;//Overload as t*
    //Tacky but effective...way to estimate attractor from S(0)
    if(use_initial_value){
           for(int row = 0; row < nActors; row++){
              Actors[row].next_v = 0.0;
              for(int j = 0; j < nActors; j++){
                    Actors[row].next_v += Actors[j].current_v * I.get(row, j);
              }
           }
    }
           return reply;
    }//Diagram_doInfluences
```

10.3 I-nets 中的冲突

上述分析假定个体之间是合作的，它们将其影响结合起来以便达成共识。但在现实生活中，社会网络充满了“竞争”关系。正和负的反馈影响导致冲突，冲突阻止 I-nets 达成一个非零的共识。忽略由 Q 给出的理论近似，实际达成共识可能是一个僵局。本节我们研究冲突，并且会发现它阻止 I-nets 达成共识。

冲突是 I-nets 中反馈循环的直接结果。I-nets 中的反馈循环既会加强也会削弱个体的状态，因为影响链路可能造成网络循环，它既可以提高也可以延缓收敛到一个非零的奇异吸引子。这些循环可能会削弱影响的效果，使网络陷入僵局——零值的奇异吸引子。当这种情况发生时，因为一个或多个个体处于冲突中，我们就说 I-nets 是冲突的。

10.3.1 冲突度

冲突度是一个阻止 I-nets 达成共识的个体的属性（通过阻止同步来进行）。正的冲突度会导致达成共识，而负的共识度会导致僵局。如果个体的冲突度是负的，则它是冲突的。当 I-nets 包含冲突的个体，所有个体的状态振荡减少直至达到零为止。这代表个体的“无立场”状态。因此，一个带有负的冲突度的个体将超越所有其他影响的效应，并使个体无法达成一个非零的态度。

考虑在图 10-4 中的 I-nets 冲突。根据理论，图 10-4 中个体节点之间影响的总和应该等于 100%：个体节点 0：24.4%，个体节点 1：55.2% 和个体节点 2：20.3%。如果可能达成共识，当节点达到接近约 24.4%、55.2%、20.3% 或这些值的组合时共识将涌现，具体要根据个体节点的初始条件而定。考虑影响度近似等于每个个体在达成决议中所控制的票数。如

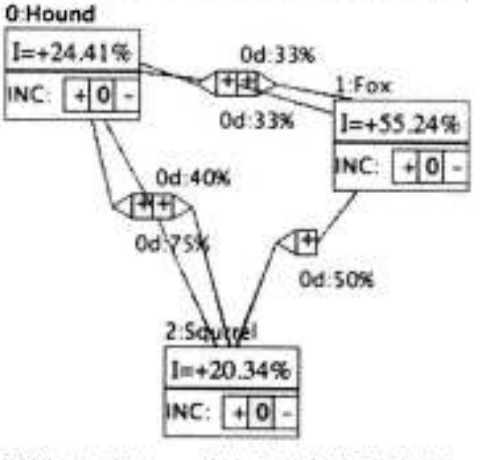


图 10-4 在 I-nets 中由于逆向循环抑制反馈而使冲突上升

果个体节点0、1和2最初设置为［1，1，－1］，I-nets应在近似（24＋55－20）＝59%时达成共识。

不幸的是，图10-4中的I-nets没有达成上述分析预测的共识值。事实上，它的奇异吸引子对除了［1，1，1］外的所有初始状态都是0。为什么？答案在于理解I-nets的反馈循环。特别是，我们应该根据反馈和前向反馈链路检查I-nets的拓扑。结果证明，这个问题类似于前面章节中研究过的基尔霍夫网络中的抑制反馈循环问题。

图10-4中冲突的I-nets中包含三个循环：在个体节点0&1和0&2之间的长度为2跳的两个小循环，以及一个较大的带有全部三个个体节点的循环。在3跳循环中，从个体节点0到节点1、2和返回节点0的累积影响，是沿着循环的影响的乘积：（1/3）（1/2）（3/4）＝1/8。我们将此指定为前向反馈循环。同样，反向循环从节点1到0和从节点2到0产生了一个反向反馈循环影响：－(1/3)(4/10)＝－4/30。将这些加起来就得到净反向反馈影响：(1/8)－(4/30)＝－(1/120)。该反向反馈循环导致整个I-nets获得的“收益”呈指数下降，阻碍了共识的达成，并最终使之下降到零。

冲突是负的影响，它是通过I-nets中的反向反馈影响实施的。结果证明，负冲突的负影响立即显现在网络的拓扑中。这种作用在系统矩阵 $\boldsymbol{C}$ 中得到。让冲突度等于 $\boldsymbol{C}=[\boldsymbol{I}+\boldsymbol{L}]$ 的对角线。$\boldsymbol{C}$ 的对角线元素 $c_i=1,2,\cdots,n$ 是由个体节点 v_i 向网络中引入的冲突度的测量。当 $c_i<0$，我们就观察到负的冲突，这对达成共识具有阻碍作用。

我们可以看到，如果 $c_i<0$，节点 $S(t)$ 的状态变化率递减：

$$|\Delta S(t)|<0 \text{ 如果存在一个个体 } v_i \text{ 使得 } c_i<0$$

因此，I-nets的奇异吸引子至少包含一个趋向于零的负冲突度。我们称对应于 $c_i<0$ 的个体节点 v_i 为冲突的，并宣布I-nets网络陷入僵局。所有冲突的个体节点必须在I－nets达成共识之前消除掉。

继续图10-4中的例子，通过从系统矩阵 $[\boldsymbol{I}+\boldsymbol{L}]$ 计算矢量 $\boldsymbol{C}$ 来确定冲突，如下：

$$\boldsymbol{\phi}=\begin{pmatrix}0 & 0.33 & 0.40\\ 0.33 & 0 & 0.50\\ 0.75 & 0 & 0\end{pmatrix}$$

$$\boldsymbol{\phi}^{\mathrm{T}}=\begin{pmatrix}0 & 0.33 & 0.75\\ 0.33 & 0 & 0\\ 0.40 & 0.50 & 0\end{pmatrix}$$

$$\boldsymbol{L}=\begin{pmatrix}-1.08 & 0.33 & 0.75\\ 0.33 & -0.33 & 0\\ 0.40 & 0.50 & -0.90\end{pmatrix}$$

$$[\boldsymbol{I}+\boldsymbol{L}]=\begin{pmatrix}-0.08 & 0.33 & 0.75\\ 0.33 & 0.67 & 0\\ 0.40 & 0.50 & 0.10\end{pmatrix}$$

$$\boldsymbol{C}=[-0.08,0.67,0.10]^{\mathrm{T}}$$

$\boldsymbol{C}$ 只有一个元素是负的，但这足以迫使共识为0。作用在个体节点0上的负的冲突度，degree_conflict(v_0) $=-0.08$，可通过改变其连接到邻接个体节点的影响的权重来加以消除。例如，在图10-4中改变了个体节点1和0之间的链路的权值，从33%至25%，产生一个没有负元素的新的冲突度矢量：

$$\boldsymbol{C}=[0,0.67,0.25]^{\mathrm{T}}$$

这种修改会消除冲突。修改后的I－nets将达成23%、53%、23%或这三个固定点值的组合

的共识，这具体取决于个体节点的初始状态。读者可以通过运行程序 Influence.jar 并从“Examples”菜单中选择这个网络来进行验证。

10.3.2 计算冲突度的 Java 方法

方法 Diagram_doConflict( )类似于近似影响度的方法，但是相对来讲没有那么复杂。它只是组成系统矩阵 $[I+L]$，然后将它的对角线上的元素转移到网络数组 Actors［i］. Actor_ conflict 中。此外，冲突的节点都被标记成红色，这显示它们之间发生冲突。如果它找到一个冲突节点，该方法就返回真，如果没找到则返回假：

```
//Find conflicting actors
public boolean Diagram_doConflict() {
        boolean reply = false;                          //Assume no conflict
        Matrix M = ConnectionMatrix();
        M = M.transpose();                              //Influences
        //Convert to Laplacian+I
        for(int row = 0; row < nActors; row++){
                double sum = 0.0;
                for(int col = 0; col < nActors; col++) sum += M.get(row, col);
                double diagonal = 1.0-sum;
                M.set(row, row, diagonal);
                Actors[row].Actor_conflict = diagonal;
                if(Actors[ row].c == 1) Actors[ row].c = 0; //Non-conflicted white
                if(diagonal < 0) {
                        reply = true;
                        Actors[row].c = 1;                  //Conflicted red
                }
        }
        return reply;
}//doConflict
```

现在我们有一个完整的 I-nets 理论体系及其建模的社会网络。如果 I-nets 的谱隙小于零，并且没有个体节点有负的冲突度，那么它将达到一个非 0 的共识。此外，I-net 的稳定性和非零共识的涌现完全就是它的拉普拉斯算子（Laplacian）矩阵的一个函数。

计算及其必须条件的总结：

$\boldsymbol{\phi}$ = 影响矩阵；$\boldsymbol{\phi}^{\mathrm{T}}$ = 影响矩阵转置

$\boldsymbol{L}$ = $\boldsymbol{\phi}^{\mathrm{T}}$ - 对角；这里对角 = $\boldsymbol{\phi}^{\mathrm{T}}$ 行的和

同步条件：$\sigma(\boldsymbol{L}) < 0$

影响度（有用性）$\boldsymbol{Q} = [I+L]^{t}$

冲突度 $\boldsymbol{C} = [\boldsymbol{I}+\boldsymbol{L}]$ 的对角

注意到很普遍的中间人或最大紧度节点的概念，在这一分析中就完全没有考虑。最强大的个体节点就是（忽略其紧度属性的）最具影响的一个。静态网络拓扑比通过该节点的路径数更重要。

10.4 命令层次结构

树形结构的社会网络或命令层次结构，是最古老的和使用最广泛的社会结构。它是由军事指挥官、公司经理和许多政府机构所使用的模型。但是，最近全球通信的改进和有见识的管理阶层引发了对命令层次价值的重新思考，特别是应用于虚拟和基于专业知识的组织。例如知识工作者在严格控制的命令层次结构中往往是无效的，引出了在现代化的互联网时代命令层次是否有效的问题。

在本节中，我们提出了这样一个问题："树形结构的命令层次作为一种管理工具是否仍然有效？"更为重要的是，"在树形结构的组织中影响力（有用性）的位置，以及什么时候命令层次可以达成共识并避免僵局？"我们证明命令层次是有效的，但自上而下对工人施加太多影响也容易造成僵局。这一结果是命令结构拓扑和对应于影响链路的权值量所直接造成的。

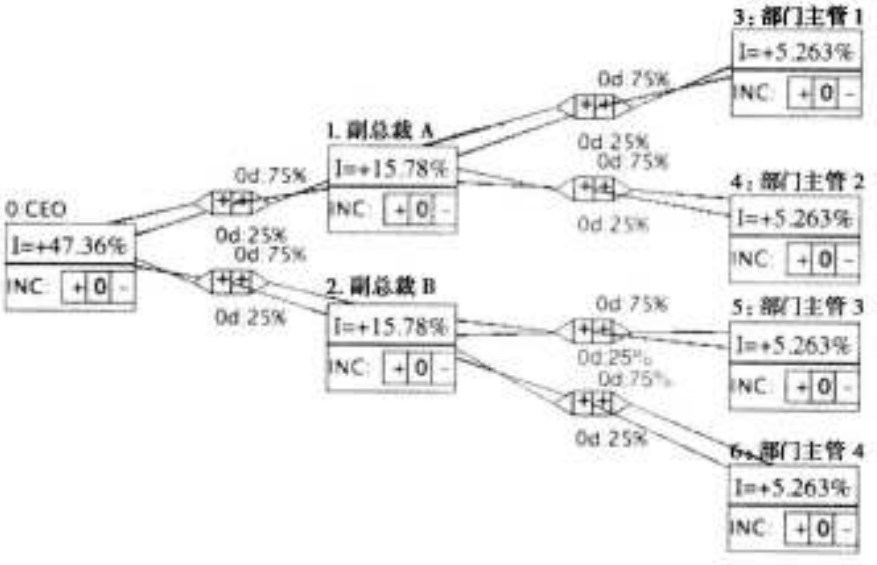


图 10-5 公司管理的典型树形结构命令层次。因为来自管理者的过分控制，这个 I-nets 是冲突的

图 10-5 展示了一个非常简单的命令层次结构，包含了一个行政总裁 CEO，两名副总裁和向每一个副总裁汇报的 4 个部门主管。这是一个简单的三层命令结构的二叉树。假设我们设定上级对下属行使高度的影响（75% 的影响），而下属对其主管（25%）行使较小度的影响，如图 10-5 所示。

人们不禁要问："行政部门经理对他或她的下属有什么影响力（权力）？"图 10-5 及表 10-3 总结了层次结构中每个个体节点所具有的影响度。只要在 I-nets 中没有任何冲突，影响的有效性就是累加的。不过，矩阵 $\boldsymbol{C} = [\boldsymbol{I} + \boldsymbol{L}]$ 的对角线分析揭示了两个内在矛盾——每个副总裁有一个冲突。表 10-3 的每一个的影响度不会达成共识，此命令层次结构会陷入僵局。不考虑图 10-5 中个体节点的初始状态，网络的奇异吸引子为零。影响权重过大，导致两个负的影响。

图 10-5 中的命令层次是过度控制的，即经理不能给予下属足够的权力。可以通过降低前向影响（即从 75% 降至 50%）来消除冲突。从本质上讲，如果他们想降低"阻止达成共识"这种冲突的话，经理必须削减对下属的影响。这种权力的共享允许共识的形成。现在 CEO（首席执行官）和两名副总裁通过多数人决定而不是发号施令来进行决策，因为他们的影响度等于 33.33%（CEO），每个副总裁的影响度分别是 16.67%。

当一个副总裁不同意 CEO 和另一个副总裁时会发生什么？根据这里形成的理论，首席执行官和副总裁的影响总和是 50%，这不够形成大多数。管理是否达成非零共识，还是网络陷入僵局？这是留给读者作为练习。

图 10-5 的简单命令层次结构表明紧密地控制命令层次容易引发冲突。事实上，如果自上而下的施压过重，则因为专制控制而使社会网络无法有效地运行。通过减少经理对下属的控制要求，就能有效地进行权力共享。然而，经理并非总是为所欲为，因为当影响度低于 50% 时，他们可能冒失去多数地位的风险。

太多的专制控制导致了低效率的组织管理，但扁平的网络组织也可以像我们在上一节中介

绍的一样过度控制。任何影响网络，无论是随机的、无标度的或小世界的，都可能由于网络中影响量和网络中反向反馈链路的性质而形成僵局。只是简单地因为它的拓扑结构，就假设任意网络组织比树形结构的命令层次（见表 10-3）更有效是不明智的。我们必须在分析中纳入影响矩阵。

表 10-3 图 10-4 的命令结构的属性

个体	链路影响	邻接个体	影响度	冲突度
0：CEO	75	1：VP A	47.36	50
1：VP A	25	0：CEO	15.78	-25
2：VP B	25	0：CEO	25	-25
1：VP A	75	3：Dept. 1	15.78	-25
1：VP A	75	4：Dept. 2	15.78	-25
2：VP B	75	5：Dept. 3	15.78	-25
2：VP B	75	6：Dept. 4	15.78	-25
3：Dept. 1	25	1：VP A	5.26	25
4：Dept. 2	25	1：VP A	5.26	25
5：Dept. 3	25	2：VP B	5.26	25
6：Dept. 4	25	2：VP B	5.26	25

10.5 I-nets 中的有用性涌现

前面讨论为社会网络提供了一个基础理论和复杂自适应系统中的依赖分析，如一组人的谈判，影响复杂经济体系的因素，重大基础设施风险评估等。但该理论并没有解释在这样的系统中个体节点的有用性（power）是如何增加的，或一个个体节点是如何变强大的。下一步是探讨需要的有用性以对整个网络的影响的形式。我们实验性地处理这一问题，使用涌现算法，通过改变进入和外出链路的影响权值提高单个个体节点的影响度。

假设我们将有用性定义为一个个体对 I-nets 中所有其他节点的影响度。影响度（因此单个个体节点的有用性）就等于其相应的矩阵 $\boldsymbol{Q}$ 上对角线上的元素。但是单个影响度是如何影响其他个体的影响度呢？为了更充分认识在 I-nets 中是如何获得有用性的，我们在接下来的两节讨论中执行两个实验：（1）获得*权重影响*——通过操作链路上的权重（影响度）有用性的涌现；（2）获得*链路影响*——通过重联链路的有用性涌现。权重影响模拟了通过说服技巧获得有用性，而链路影响指大家熟悉的“人际关系的”，即被认为是通过“你认识谁而不是你知道什么”而受益的工人。

我们发现改变进入和外出链路的权重来获得有用性会导致少量的个体节点对所有其他节点进行控制。具体来说，在 I-nets 中个体节点获取有用性的最好策略是减少进入链路的权重，并增加外出链路的权重。对社会网络分析而言，建议单个个体节点忽视来自邻近的个体节点的意见，并侧重于增加对其他个体节点的影响。对人类来说，一个有说服力的外向的人比一个有说服力的内向的人一般能够获得更多的有用性！事实上，我们证明了这一策略大部分的时间会导致 I-nets 的独裁。

我们在链路影响涌现中发现了相反的结果——I-nets 中重联链路对获得有用性收效甚微。事实上，我们发现链路影响涌现会导致混沌，并且就获得有用性而言没有可以使用的工具。这种负面的结果是重要的，因为它经常假设有用性可以通过与其他强大的个体节点联系起来而获得。然而，我们发现这是错误的，至少对于这里建立的模型如此。

10.5.1 加权涌现

加权涌现的总体思路在于增加外出链路的权重来增加一个个体节点作用于外出邻居的影响，减少进入链路的权重以最小化其他个体节点作用于（强大的）个体节点的影响。从影响矩阵 $\boldsymbol{\phi}$ 来看，这种方法是有意义的。矩阵 $\boldsymbol{\phi}$ 中行 i 代表个体节点 i 对其他个体节点的影响，列 j 代表其他个体节点作用于个体节点 j 的影响。显然增加行 i 的总数以及减少列 i 的总数对个体节点 i 有两个好处：（1）它增加了节点 i 作用于邻接节点的总的影响；（2）它降低了其他个体节点作用于节点 i 的总的影响。两种行动都增加个体节点 i 的影响度。

矩阵 $\boldsymbol{\phi}$ 中增加行的总数同时减少列的总数的问题是很明确的——这些都是相反的行动。减少列总数也减少了行总数，反之，增加行总数也增加列总数。我们不能同时进行两种操作，因此所建议的策略在数学上毫无意义。

不用数学方法解决这个矛盾，I-nets 通过涌现过程为我们找到了解。作为对书写方程式的替代，我们建议编写每个个体节点都可以独立遵循的微规则，然后从数百微规则的应用中观察到宏观结果的涌现。I-nets 即可以收敛也可以发散。如果它收敛，（部分）稳定节点的影响度就能回答问题“谁是 I-nets 中最强大的个体节点?”

加权涌现算法很简单：从影响矩阵 $\boldsymbol{\phi}$ 中选择两（行，列）对，对应于个体#行的一个进入和外出链路。然后从进入链路减去一点权值，并增加一点到外出链路上。如果条件是对的，逐步改变进入、外出链路的权值导致了在随机选取的个体节点上影响度的增加，但如果是错的，我们恢复到以前的权值。这个过程重复直到所有个体节点的影响度依然（相对）稳定，但或许相互不同。

对于这种涌现工作的正确条件是什么？有三个条件需要考虑：（1）矩阵 $\boldsymbol{\phi}$ 的列和行的总和不要超过 100%，也不要让它们下降到 0%；（2）影响（链路上的权值）的 1% 变化不能在 I-nets 中创建冲突节点；（3）选择的个体节点的影响度一定不能减少。以下启发式将满足所有三个条件。

加权涌现

1. 随机选择一条链路连接个体节点#row_1 到进入的个体节点，第二条链路到外出个体节点：

$$\text{外出链路：}\ \text{row}_1 \rightarrow \text{col}_1$$
$$\text{进入链路：}\ \text{row}_2 \rightarrow \text{col}_2$$

2. 从影响矩阵 $\boldsymbol{\phi}$ 中通过行和列的总数计算进入和外出的总数；

$$\text{进入总数} = \sum_r \boldsymbol{\phi}(r, \text{col}_2)$$
$$\text{外出总数} = \sum_c \boldsymbol{\phi}(\text{row}_1, c)$$

3. 保存进入和外出链路权值的副本，以防条件不正确时改变它们。
4. 如果外出总数≤99%，外出链路权值≤99%，增加外出链路权值 1%。
5. 如果进入总数≥1%，并且进入链路权值≥1%，减少进入链路权值 1%。
6. 重新计算影响，$\boldsymbol{Q} = [\boldsymbol{I} + \boldsymbol{L}]^t$，并检查冲突。
7. 如果个体节点#row_1 的影响减少或发现了冲突，就改回到存储的权值。

外出链路的权值只能增加，而进入链路权值只能减少。因此，外出链路获得更多的影响，而进入链路失去影响。但一个个体节点的进入链路是另外一个个体节点的外出链路。提高和降低进入和外出链路的权值似乎形成了一个悖论。这个悖论引起了一个问题，“这种涌现过程是否会导致稳定的模式，还是会导致失控的振荡?”如果个体节点有严格相同的进入和外出链路度，我们会得到不受控制的振荡，但总的来讲，I-nets 是不对称的，也就是说，度序列并不均匀一致。这表明，外出链路比进入链路多的个体节点要比外出链路比进入链路少的个体节点更强。如果

条件是正确的，越强的个体节点就应该获得更强的有用性，因为它的外出链路获得的比进入链路失去的影响更多。这正是从加权涌现过程中涌现的。然而，基本的假设在于，收敛 I-nets 的度序列是不对称的，或者至少是不均匀的。

10.5.2 加权涌现的 Java 方法

链路权值是由方法 Diagram_doWeightEmergence( ) 更改的，这由程序 Influence.jar 重复数百个时间步来实现。该算法在概念上是简单的，但在执行时有些混乱，因为它必须操纵影响矩阵，检查三个必要的条件以保持网络稳定，并将行列坐标映射对应到链路索引。

影响矩阵 ϕ 是涌现操作的核心，它存储在方法的矩阵 W 中。大多数微规则由 W 操作来执行，转换 W 的［行，列 （r，c）］坐标到影响数组的链路索引是有必要的。转换工作由方法 connection(r,c) 来执行，将矩阵坐标 （r，c） 翻译成链路索引：

```
private int connection(int from, int to){        //Map (from, to) -> link#
     int reply = -1;                             //Return link index or (-1)
     for(int i = 0; i < nInfluences; i++){
         if(Influences[ i] .from == from && Influences[ i] .to == to) {
            reply = i;
            break;                               //Found
            }
         }
        return reply;
  }//connection
```

方法 Diagram_doWeightEmergence( ) 也必须保证随机选择的进入和外出链路是独一无二的，并且不对应于矩阵 W 中的同一元素。如果碰巧随机选择的两条链路是相同的，就进行另一个随机链路的选择。count 变量用来保证用于寻找唯一链路的 while 循环终止。如果不符合这一条件就会立即从该方法中退出。

下一个主要步骤是保证涌现过程中收敛（若可能的话）的三个必要条件中的第一个：列和行总数不超过 100%。进入和外出的影响是通过 W 适当的行和列的总和来总体计算的。核对这些总和不超过 1% 和 99% 的界限。请注意使用 W. get 和 W. put 操作 W 的元素。这些访问方法由 JAMA 软件包来定义，用于执行矩阵类操作。

第二个条件，即没有冲突的节点，通过调用 Boolean 方法 Diagram_doConflict( ) 来评估 I-nets 的冲突是否满足。如果有冲突，改变的影响被改回原来的权值。回顾一下，只需要一个冲突个体节点就会使 I-nets 不稳定。

第三个条件是通过调用方法 Diagram_doInfluences(false) 来实现的，该方法计算个体节点影响度的变化，作为改变进入和外出链路的结果。如果随机选择的个体节点的影响已经下降，改变的影响会被重置为先前的权值。只要用户允许，重复应用这种方法。通常情况下，涌现过程开始遇到冲突就会停止。但是正如实验所示，并不总是如此。

```
public void Diagram_doWeightEmergence(){
    Matrix W = ConnectionMatrix();
    int row1 = (int)(nActors*Math.random());   //Random actor #row1
    int col1 = (int)(nActors*Math.random());
    int out_link = connection(row1, col1);
    int count = 100;
    while(count > 0 && (row1 == col1          //Distinct, actual out-link
            ||
       out_link < 0)){
```

```
        row1 = (int)(nActors*Math.random());
        out_link = connection(row1, col1);      //Map to link #
        count--;                                //Avoid infinite loop
      }
      if(count == 0) return;                    //Out-link doesn't exist
      int col2 = (int)(nActors*Math.random());
      int in_link = connection(col2, row1);
      count = 100;
      while(count > 0 && (row1 == col2          //Distinct, actual in_link
          ||
        in_link < 0)){
        col2 = (int)(nActors*Math.random());
        in_link = connection(col2, row1);             //Map to link #
        count--;
      }     .
      if(count == 0) return;                          //In-link doesn't exist

      //Sum in-link, out-link influences
      double in_sum = 0.0;
      for(int row = 0; row < nActors; row++)
        in_sum += W.get(row, col2);

      double out_sum = 0.0;
      for(int col = 0; col < nActors; col++)
        out_sum += W.get(row1, col);

      //Save for reverting, if necessary
      double save_out_link_weight = Influences[out_link].weight;
      double save_in_link_weight = Influences[in_link].weight;

      if(out_sum <= 0.99 && Influences[out_link].weight < 99)
        Influences[out_link].weight += 1;
      if(in_sum >= 0.01 && Influences[in_link].weight > 1)
        Influences[in_link].weight -= 1;

      //Revert if conflicted or decrease in influence
      global.Message = "Weight Emergence: ";
      boolean conflicted = Diagram_doConflict();
      if(conflicted)
        global.Message += "Conflicted at Actor #"+Long.toString(row1);

      double before = Actors[row1].Actor_influence;     //Save current influence
      Diagram_doInfluences(false);                      //Has influence improved?

      if(before < Actors[row1].Actor_influence || conflicted){
        Influences[out_link].weight = save_out_link_weight;
        Influences[in_link].weight = save_in_link_weight;
        global.Message += " Reverting.";
      }
      else
        global.Message += " Increasing Actor #"+Long.toString(row1)+
        ",Influence = "+s0.toString(100*Actors[row1].Actor_influence)+"%";
    }//Diagram_doWeightEmergence
```

请注意该算法是如何允许一个或两个随机选择的链路改变权值的。具体来讲，进入的影响可以减少1%，不考虑外出的影响是否增加了1%。因此，涌现过程是一系列并非真正小的交换，而是进入影响的微小减少和外出影响的略微增加。直觉上，似乎这个过程总是导致零权值的进入影响和100%权值的外出影响，但实际上会更为复杂。

10.5.3 加权涌现的稳定性

我们使用程序 Influence.jar 来研究加权涌现的稳定性，并回答上一节提出的一些问题。下面的实验测试收敛部分得到保证的假设，最强大的个体节点最终获得最大影响度。我们说有用性是直接与分配到外出链路的影响的数量相关的。进入链路减少了个体节点的有用性，因此它们应该最少化。外出链路比进入链路多的节点是“最强大的”，同时进入链路多的节点是“最弱的”。我们通过模拟来测试这个假设。

在加权涌现的仿真过程中，如果发生冲突，个体节点就标记为红色，如果外出链路比进入链路多则标记为蓝色。当个体节点达到用红色表示的冲突状态时，我们期望这一过程收敛，而且我们期望蓝色个体节点能够以最大影响度来结束。如果它们这样做，那么我们有经验证据表明，这一假设是正确的。然而，我们告诫读者，这种经验与数学证明是不相匹配的！我们把正式的证明留给读者作为练习。

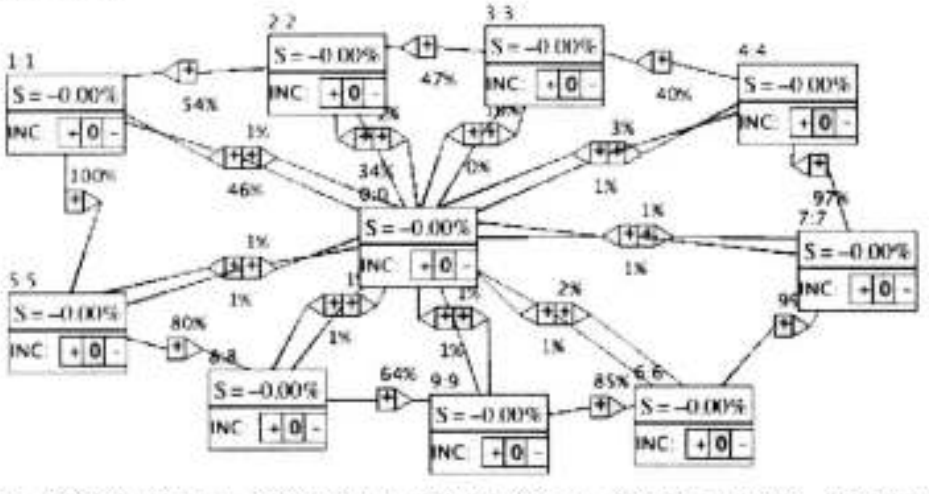


图10-6 辐射状星形 I-nets，边沿个体节点围绕节点0构成一个无向的环状网络，并且节点0通过双向链路链接到其他每个节点上

图10-6的辐射状星形（spoke and hub）网络用于执行实验并收集结果。最初所有链路都分配了一个小的权值10%。方法 Diagram_ doWeight Emergence( )反复调用直到冲突水平达到某一点（在该点处）单个个体节点上的影响即使允许该方法增加也是微乎其微的。这标志着收敛近似于某一稳定状态。但是，细心的读者会发现，影响网络从来没有达到一个奇异吸引子，因此不会完全同步。涌现以一种偏差的模式更改权重。通常，模式以一个或多个得到大多数影响（即使不是全部）的主导个体的形式涌现[⊖]。

加权涌现在大多数时候显示出部分稳定性，带有更少进入链路的节点控制着带有更多进入链路的节点。个体节点的外出链路比进入链路多，我们称这种个体节点为主导个体，并假设它们最终获得整个网络的最多的有用性。程序 Influence.jar 标记这些节点为蓝色，表明它们可能是网络中的主导个体。在实践中，最终实现最有影响的个体节点是部分偶然的结果，随机选择过程的

⊖ 主导被定义成“高度影响”并且经常与其外出链路比进入链路更多的节点关联

结果是有利于早前在涌现过程中选择的个体节点。

图 10-7 展示了进入和外出链路数对影响度的影响，因此一个个体节点的有用性控制着包含 10 个个体的 I-nets。使用了两种拓扑图获取图 10-7 的数据：（1）如图 10-6 所示的纯辐射状星形拓扑；（2）一个类似于图 10-6 所示的纯环状拓扑，但有一个周边节点作为 hub。环状拓扑结构是通过消除图 10-6 中节点 1 和 5 之间的链路获得的。这已经创造了一个额外的占据主导的个体与 hub 竞争的效应。如预计的一般，竞争者将从 hub 上剥夺一些影响。

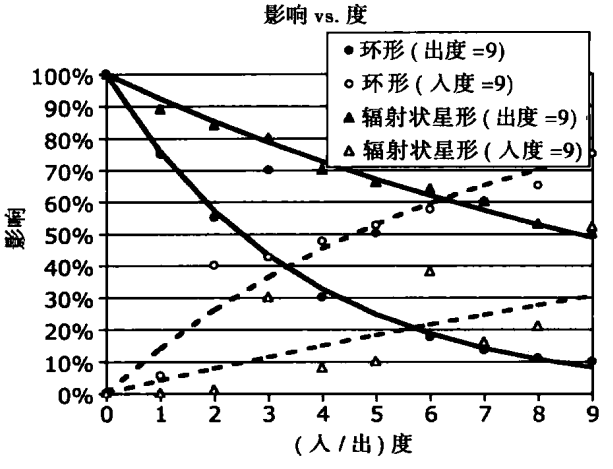


图 10-7 图 10-6 中最占据主导的个体节点的影响度随着其外出“辐条”数目的增加按指数增长；而影响度随着其进入“辐条”数目的增加按指数下降

执行了两个实验：一个是外出链路的数量固定在 9，而进入链路数量可在 0 到 9 之间变化；另一个是进入链路的数量固定在 9，而外出链路数量可在 0 到 9 之间变化。这两个实验产生的两组数据对应于如图 10-7 所示的两对曲线。

纯环形拓扑的结果（相对应“环形”标记或实线）显示影响度随着进入链路数从 0 到 9 增加而外出链路数量不变时呈指数下降。图 10-7 的虚线显示影响随着外出链路数上升而进入链路数量不变时呈渐近指数增加。类似的但不太明显，在辐射状星形的拓扑中发现了指数关系。

这个实验提供了经验证据以支持假设。至少对于 I-nets 的这种配对，增加外出影响链路的数量会增加个体的有用性，同时增加进入影响链路的数量会减少有用性。具体来讲，图 10-7 中的数据拟合指数关系，实线要比虚线拟合得更好：

出度 =9

Actor_power(in_degree, 9) = exp(−0.28(in_degree)), 对于环形

Actor_power(in_degree, 9) = exp(−0.08(in_degree)), 对于辐射状星形

入度 =9

Actor_power(9, out_degree) = 1 − exp(−0.15(in_degree)), 对于环形

Actor_power(9, out_degree) = 1 − exp(−0.04(in_degree)), 对于辐射状星形

图 10-7 的曲线拟合对外出链路的数量变化比进入链路更为准确，这使我们猜想在 I-nets 中有用性与外出链路的数量密切相关，与进入链路的数量弱相关。额外的外出链路可以增加有用性，而进入链路的加入可以抑制有用性。在社会网络中，说服其他人按照你的思维方式比被别人说服更好！

当两个或更多的个体节点有相同数目的外出链路时会发生什么？如果两个个体节点通过一个有向链路联系在一起，那么“to actor”受到“from actor”的影响，但不是相反。外出链路占据主导，这意味着“from actor”更为强大。

如果两个个体的连接由等权值的外出和进入影响，它们会相互抵消，也就是说，两者是紧密耦合的，如图 10-2 所示。此外，如果两个占据主导的个体互相抵消，第三个个体节点可以控制它们和网络，即使它有比外出链路更多的进入链路也是如此！强大的节点相互阻塞所留下的权力（有用性）真空，可以由其他节点所填补。为什么呢？这个问题留作读者的练习。

10.5.4 链路涌现

在 I-nets 中个体节点通过忽略进入影响并增加外出影响的权值来获得有用性，但它们可以通过将链路切换指向到更强大的相邻节点来获得有用性吗？链路涌现的思想很简单，随机地重联链路类似于创建无标度网络的偏好连接。链路涌现不是选择高度节点，而是选择具有高影响度的节点。直观地讲，个体节点的影响应该随着它影响更强大的邻居而增加。但是我们发现结果并非如此！

程序 Influence.jar 实现链路影响涌现如下。随机地选择一个个体节点和它的外出链路之一。如果重联外出链路（以便于它将指向具有更大影响的邻接节点）增加个体节点的影响度，则保持新链路；否则恢复到以前的连接。链路上的权值并没有改变，改变的只是它们的连接。

我们假设链路偏爱具有高影响度的个体会引起 I-nets 演变成一个有少量高连通、高影响的个体节点和大量单独的弱个体节点的网络。但是，我们从前一节中知道进入链路削弱了影响而不是加强了影响。事实上，强大的个体节点被太多的进入链路“拖累了”。再次，我们面对矛盾现象，与强大的个体节点关联的弱个体节点降低了强大节点的有用性。随着有用性的削弱，链路涌现重联链路并远离弱节点，导致随后它们的有用性增加！链路涌现会导致混沌！

练习

10.1 二叉树 buzz 网络的吸引子值可能是什么？

10.2 设计开发一种理论并解释 buzz 网络的线性行为：线性网络可能的吸引子值是多少？

10.3 buzz 网络的吸引子值同促销者和反对者的度之间的数学关系是什么（如果有的话）？

10.4 如果链路的权值从 33% 改为 20%、50% 改为 40%、75% 改为 60%，图 10-3 中节点 0、1 和 2 的影响度是多少？如果 $S(0) = |+1, 0, -1|$，那么 I-nets 网络达成的共识是什么？

10.5 图 10-4 中 I-nets 网络是冲突的。如果影响从 1：fox（狐狸）到 0：hound（猎狗）由 33% 降低至 25% 或 26%，它仍然是冲突的吗？

10.6 假设图 10-5 中的前向影响链路降低至 50%，而不是 75%。如图中反向反馈链路保持在 25%。当 CEO 和副总裁 A 同意（+1）、副总裁 B 不同意（-1）时，二叉树达成的共识是什么？如果副总裁 B 是中立的（0），达成共识的值又是多少？

10.7 图 10-8 中的 I-nets 网络证实了对于外出链路比进入链路多的个体占据优势规则的例外。在权值涌现后哪个节点是最强大的个体节点？解释为什么外出链路比进入链路多的个体节点不是最强大的个体。

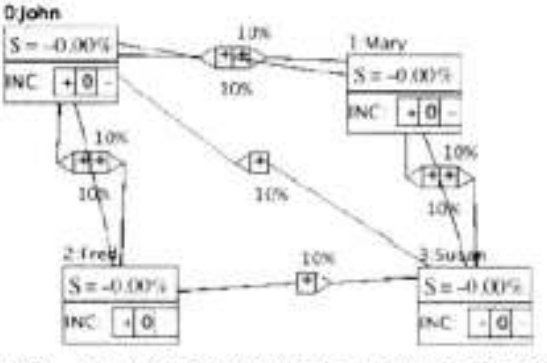


图 10-8 反例：主导个体要比其他个体有更多外出链路是规则的例外

10.8 考虑如图 10-9 中所示的宏观经济学的 I-nets 网络模型。当所有的影响为 20% 时，哪一个个体具有最高的影响度？经过加权涌现后哪个个体上升到最强的位置？

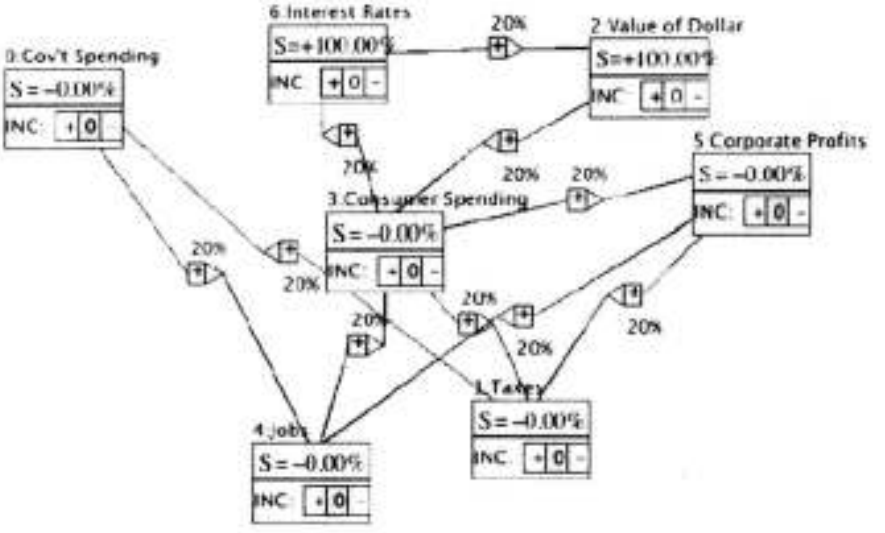


图 10-9 I-nets 网络形式的国家经济的简单宏观经济学模型

10.9 ［研究］解释加权涌现应用到简单线形网络上的结果。你的线形 I-nets 网络不服从“最强大的节点其外出链路比进入链路更多”规则吗？为什么？

10.10 ［研究］设计一套完备理论，并说明在 I-nets 中的个体变成最强大的个体的必要和充分条件，与进入链路和外出链路数无关。

脆 弱 性

尽管有许多定义脆弱性、风险及后果的方法，但很少有文献提供有关网络的脆弱性、风险及故障后果的定义。这是因为网络科学是一个新的领域，而且并不清楚网络是如何发生故障的。举例来说，当某个链路被毁掉时，该网络是否会发生故障？节点是否要比链路更关键？我们大体上将网络脆弱性沿着三个方面分类：脆弱性由于（1）一个或多个节点的故障，（2）一条或多条链路故障，（3）由于删除一条稳定链路导致的不稳定的（基尔霍夫）网络。因此，节点和链路都是导致不同种类故障的原因。首先，我们研究节点损坏的影响，然后是链路损坏的影响。

进一步来讲，网络中节点或链路故障所产生的后果并没有得到很好理解。对于一个网络，是否有必要分成独立的组件以便在故障时对它进行分类？换一种方式来讲，在一个支持流的动态网络（如基尔霍夫网络）中，故障就可以定义成流的停止吗？最后，故障可被定义为在一个同步的网络中因为某个链路的拆除而不再同步吗？

弹性的概念加剧了这种理解上的混乱，因为对网络弹性没有标准的定义。我们如何比较两个网络的弹性？直观上讲，由于节点和链路被删除，弹性与该网络中剩余的实用功能总量成正比。这里所追求的一个实实在在的思想是对节点删除及链路删除可以应用不同的弹性度量。例如，我们可以探讨网络中节点删除及链路删除后的承受力，以及在基尔霍夫网络中通过有向网络的最大流量。在这种限制的情况下，弹性仅是通过网络的有价值的物品的最大流量的一小部分，它作为衡量节点及链路可靠性的一个函数。

关于非网络系统中由什么来构成脆弱性、风险及其后果，金融和工程界有一个共识。我们将这些称为标准定义。脆弱性 V 是某个部件或重要资产遭到攻击后发生故障的概率。风险 R 是由于故障所期望的损失，威胁 T 是一个攻击尝试的概率。成功的攻击产生后果 C，这是由攻击造成的损害。因此，标准风险简洁定义成乘积：$R = TVC$。

标准威胁，脆弱性和风险

T = 尝试攻击的概率

V = 尝试攻击成功的概率

C = 由成功攻击造成的后果及危害

R = 标准风险，定义为乘积 TVC

在本章中，我们将这些标准定义扩展到包含许多组件或资产（节点和链路）的网络上。网络威胁、脆弱性、后果和风险将是各个组件或资产的威胁、脆弱性、后果和风险的总和。假设一个组件既可以是一个节点也可以是一条链路，我们将网络风险定义为单个组件的所有节点和链路的总和。

网络威胁，脆弱性和风险

t_i = 组件 i 受到攻击的概率，为了不失一般性，我们假定 $t_i = 1$，因为 v_i 出现时，如果 $t_i \neq 1$，我们就可以替换成 $t_i v_i$

v_i = 攻击组件 i 成功的概率

c_i = 对某个节点/链路 i 成功攻击造成的危害及后果

$r_i = t_i v_i c_i$：单个组件的风险

$R = \sum_{i=1}^{n+m} r_i = \sum_{i=1}^{n+m} t_i v_i c_i \sum_{i=1}^{n+m} v_i c_i$，因为我们假定 $t_i = 1$

我们会发现通过将风险值缩放到［0，1］区间来标准化是很方便的。在这种情况下，标准化的风险值是最差情况下的风险的比率（$v_i = 1$）：

$$R_{\text{norm}} = \frac{R}{R(1)} = \frac{\sum_{i=1}^{n+m} v_i c_i}{\sum_{i=1}^{n+m} c_i}$$

这些都是可能的最基本定义，而且它们很容易应用。总的来讲，对于任何系统（其中故障概率 v_i 的先验近似可以被合理估计）中的风险，它们形成了概率风险分析（PRA）方程。然而，我们将 PRA 的定义扩展，以容纳适应遭受攻击的网络拓扑。事实上，我们问了两个基本问题，“网络中风险的实质是什么，以及防护人员如何应用有限的资源减少网络风险?”具体来讲，我们研究对网络节点和链路攻击时的风险。我们说明了保护网络不受对链路和节点攻击的最优策略。

在本章中，我们将学习：

1. 网络的损坏是由意外或自然作用（随机节点/链路的拆除）或有针对性地攻击节点和链路（犯罪和恐怖行为）所造成的。我们证明了存在最佳的攻击策略，它直接与网络拓扑结构相关。例如，删除高度 hub 比删除低度 hub 会对网络产生更大的破坏。

2. 网络风险值定义为 $R = \sum_{i=1}^{n+m} t_i v_i c_i$，其中威胁 t = 攻击的概率，脆弱性 v = 企图攻击成功的概率，后果 c = 一个攻击成功后发生的后果或损害。但函数 $\Phi = \sum_{i=1}^{n+m} g_i v_i c_i$ 作为一个优化目标函数会更有用，因为它包含了网络的度序列分布。这使得将网络的拓扑结构与风险属性关联起来成为可能。

3. 网络风险的策略与资源分配策略相当，通过优化资源分配“买断”风险；通过各种保护措施使节点/链路的风险降低也就意味着降低了脆弱性。我们在两种重要情况下求解了网络的最佳资源分配问题：线性成本和指数成本模型。

4. 关键节点/链路是具有最高 $gC/\max DA$ 值的资产，其中对链路而言 $g = 1$，对节点而言 $g = \text{degree}(v)$；C = 后果；脆弱性降低成本（$\max DA$）= 删除大部分或全部脆弱性需要的资源。我们证明了最优分配策略服从由 $gC/\max DA$ 建立的排序属性，而与脆弱性和资源分配 DA 之间的关系无关。

5. 关键节点和链路被定义为具有最高 $gC/\max DA$ 值的节点/链路。最好的资源分配策略会为关键节点/链路分配更多的资源。

6. 我们研究了 Al-Mannai 和 Lewis 的攻击 - 防御模型，显示出攻击者的最佳策略是不对称的，即攻击者与防御者资源分配在直径上刚好相对（截然相反），攻击者应该为不关键的节点和链路分配比关键节点和链路更多的资源。防御者应将大部分资源分配到关键节点/链路上，而在不关键的节点/链路上分配较少的资源。这一结果是在假定攻击者和防御者了解对方的策略和资源分配的情况下得出的。

7. 我们证明对于不了解攻击者分配策略的防御者来讲线性分配策略是最好的，而指数分配策略则最适用于攻击者。线性策略使得防御者风险最小化，而指数策略使攻击者的风险最大化。

8. 链路弹性被定义为将网络分成单独的组件所必须删除的链路部分（分数）。我们说明小世界模型是最有链路弹性的，随机网络其次，而无标度网络是最缺乏弹性的。换言之，删除更少的

链路就会使无标度网络分隔成组件，而随机或小世界类的网络则不会。

9. 流弹性被定义为当链路阻塞或删除时有向网络所支持的最大流量的分数。我们研究了一种启发式优化分配，分配资源保护流网络不会由于沿着关键路径上的链路的删除而产生故障，根据有效的度量 $\min C_i/\max DA_i$ 排序。

10. 删除一个或多个关键链路会使基尔霍夫网络不稳定。这种基尔霍夫网络类是无法稳定的，除非其反馈回路（或称反馈循环）的长度相对都是质数。关键链路是指那些如果删除了会导致基尔霍夫网络不稳定的链路。这是因为删除关键链路会更改反馈回路的长度，即删除了一个长度相对于另外一个循环来说为质数的循环。稳定性弹性定义成对稳定性不关键的链路部分。

11.1 网络风险

与单一资产攻击相比，网络攻击的概念并不再是新鲜事。实质上就是仅通过删除某个组件就会中断或摧毁整个系统的策略。例如，二战期间盟军轰炸机摧毁德国唯一生产球轴承的工厂，因为知道这会造成无数的战争机器如飞机、坦克、运兵车及舰船等生产的停顿，因为球轴承在几乎所有类型的机器中都会用到。盟军熟知系统及其对关键零部件的依赖性。在这种情况下，德国球轴承工厂正好是所有其他更大系统必不可少的一个组成部分。

网络科学是建模构造复杂的相互依存的系统（如关键的基础设施系统、制造业逻辑斯蒂链和生产流程）的一种优秀的载体。Lewis 使用了以下描述的技术分析了一系列的电网、供水系统、电信系统、能源分布网络和交通系统（Lewis, 2006）。他认为，基础设施网络的确是有结构的，并且电网拓扑结构、电信网络和互联网，通过只保护少数的节点和链路（即所谓的关键节点和链路）就可以有效地加以保护。很多其他研究人员通过探讨删除节点和链路网络的生存效应来研究这一问题。然而，对于网络的生存能力、脆弱性、可靠性及弹性的定义还没有一致的定义。我们接下来阐述这一主题。

由于其抽象的本性，网络可以模拟几乎所有可分解成组件（节点）及它们之间的关系（链路）的系统。例如，德国的战争机器系统可以表示成一个工厂系统（节点）和供应线路（链路）。显然，某一节点依靠其他节点供应，其他节点取决于上游的另外节点等，直到代表原料来源的节点为止。供应链链路显示了材料的运动以及从原材料到半成品再到成品的过程。从这个意义上讲，供应链是一个模拟了某种商品流或从源头流向接收节点的商品的网络。

网络可能代表原料的流向，沿着生产链的步骤或控制流（包括人工和自动化）导致对一些商品的生产或决策。我们提出的问题是，“如何才能使这样的一个系统中断或停止?”此外，应考虑什么样的中断措施？在一般情况下，摧毁或禁用一个或多个其节点或链路来中断网络。相反，我们可以通过保护或强化节点或链路来保护网络，或两者同时兼而有之。假如基于强化每个节点或链路代价太高，那么在规定预算下采用何种最佳策略以尽量减少损失呢？这个问题的回答是通过制定和求解一系列资源优化方程式，找出现有预算约束下的最佳分配来尽可能强化节点和链路。

首先，我们探讨了基于节点的网络保护问题——从而产生了关键节点分析。然后我们探讨了通过保护其链路来保护网络的问题——从而产生了关键链路分析。最后，我们介绍了网络的脆弱性的新概念——在基尔霍夫网络中删除链路会造成不稳定。

网络很容易受到对节点和链路的威胁，但威胁是什么，一个网络容易受到威胁时又意味着什么？单一资产如工厂（节点）的脆弱性和系统如逻辑斯蒂供应链（网络）的脆弱性之间有什么区别？我们将威胁定义成一种特定形式的攻击，如在互联网上使计算机瘫痪的网络蠕虫，或用来摧毁建筑物的炸弹。我们进一步将脆弱性定义为一种衡量系统弱点的方法。例如，在通信网络中的计算机由于防火墙的弱点可能容易受计算机病毒或网络蠕虫的攻击。最后，将由一次成功

的攻击造成的后果定义为时间上、金钱上甚至生命的损失。风险是由于对一个系统的成功攻击的预期损失。

假设脆弱性和威胁以概率形式表示，风险为由于威胁和脆弱性的预期损失。下面的公式定义通用于本章：

T = 威胁：（通常是通过特定武器的）一次攻击的概率

V = 脆弱性：一次成功攻击的概率

C = 后果：一次成功攻击所造成的损失

R = 风险：预期损失，$R = TVC$

这些都是众所周知的应用于概率风险分析（PRA）中的工程和财务分析定义。PRA 将风险定义为一个预期值——TVC，其中 T 取决于攻击的性质（炸弹、网络蠕虫或生物传染）；V 是一个组件或资产如果受到攻击或压制而产生故障的可能性，它也依赖于攻击的性质；C 是如果发生（财务上或致命的）故障时的后果。威胁和脆弱性是对故障概率的一种先验估计。后果 C 一般以金钱或生命代价来衡量。

简单的 PRA 的定义必须扩展以便计算系统中的风险，因为系统故障也是节点的相互依赖性的函数，也就是说，当计算风险时就与网络连通性有关。德国在二战期间唯一的球轴承工厂的故障造成了严重的后果，因为其关联到许多其他工厂。因此，我们需要一个扩展定义表示系统组件风险，也就是说，我们需要能用于网络的风险定义。在下面的分析中，整个网络上的单个节点/链路风险加起来就得出一个总的网络风险。

11.1.1 将节点作为目标

首先，考虑风险与受损节点相关。系统的损失是网络连通性的损失以及单一节点本身价值损失的结合。直观地说，一个节点的目标价值应包括其内在的价值以及它对整个网络的价值。PRA 风险定义的一个简单合乎逻辑的扩展是将节点的目标价值定义为 $g_i C_i$，其中 g_i 是节点的度，C_i 是与节点内在价值相关联的后果。一个节点如果有两倍的链路将会有两倍的价值，因为它的删除会造成网络的双倍损害。

Lewis 和 Al-Mannai 使用网络风险的度加权模型来确定网络中哪个节点对网络的整体安全最重要（Al-Mannai，2007）。具有最高风险分布的节点 $r_i = g_i C_i$，对整体网络的安全性是最重要的，Al-Mannai-Lewis 方法被称为关键节点分析。Lewis 将关键节点分析应用于各种关键的基础设施中，如水、电、能源、交通运输和电信系统，并说明了在哪里可以找到最重要的故障点（Lewis，2006）。关键节点分析在本章后面会更详细地探讨。

Lai、Motter 和 Nishikawa 研究了网络中层叠的故障节点效应（Lai，2004）。网络层叠是从一个单一节点/链路故障开始像传染病一样传播到连接的节点的一系列故障。Lai 等人用负载 L_i 和偏差参数 $\alpha \geqslant 0$ 模拟了节点 w_i 的目标值。参数 α 可被视为一个冗余因素，当主节点故障时就会提供冗余容量。节点 w_i 故障后果与它的负载和冗余参数有关：

$$C_i = (1 + \alpha)L_i, i = 1, 2, \cdots, n$$

Lai 等人说明了最关键的节点是具有最高负载的节点，例如，最大的后果。此外，当负载分布不均匀即从节点到节点负载差异较大时，网络对故障就更加脆弱。该效应的解释是，网络由于高负荷节点故障而向其他（邻近）节点传播所增加的负荷更为困难所造成的。这种困难会随不均匀性增加而增加。

随着链路数量的增加，节点上的负荷也会增加。因此，当网络变得无标度时，Lai 等人的模型会变得更像 Al-Mannai-Lewis 模型。其实，hub 对整体网络的风险贡献更大，因为删除 hub 也会禁用大部分链路。事实上，Albert 和 Barabasi 称 hub 为网络的死穴，因为它们在网络的破坏方面是最重要的节点（Albert，2000）。Lewis 通过仿真模拟结果显示，由于层叠而导致整个网络彻底

故障的概率，会随着目标节点度的增加而显著增加（Lewis，2006）。如果防御者资源有限，最好的策略是保护最高度的节点。这一结论假设基于所有节点与链路都有同等价值，网络是同质均匀的。当后果因节点和链路的不同而变化时，这一结果并不成立——异构网络的防御依赖于后果和强化节点的费用及度序列。

我们得出结论，如果随机选择节点目标，均匀的无标度网络要比随机网络更有容忍性，但如果目标针对 hub 时则恰恰相反。由于它们的高连通性，hub 是所有节点中最关键的，而低度的节点则不是那么重要。但这一结论是在假定所有节点都具有同样价值的情况下得出的。当将任一高价值的节点作为目标时，我们必须考虑连通性和节点的后果值。这将导致随后更正式的分析——关键节点分析。

11.1.2 将链路作为目标

链路对于网络的运作也很重要，特别是如果用网络表示一个包含某些商品流的系统时更是如此。Albert 和 Barabasi 研究了从网络中随机地删除链路，直到巨大的连通组件网络被一分为二为止。但是，在现实世界中，随机地删除具有同样目标价值的链路毫无意义。因此，Lai 等人研究了对具有相等目标价值的完整小世界和无标度网络中链路的删除效应。再次证明，最重负载的链路要比轻负载的更为重要。Lai 等人在具有小世界效应网络中还发现了一个与直觉相反的结果——在无标度网络中，短程链路要比远距离链路更容易受到攻击。回想一下，远程链路是有助于小世界效应的将网络二分的捷径。为什么本地链路要比捷径链路更关键？

但是 Lai 的违反直觉的结果并不通用，在由 Lai 等人进行的仿真中，这是由于短距离（本地）的链路承担了最大载荷所造成的后果。这一结果完全证实了正确的直觉暗示，拥有更高目标价值的链路更关键。因此，在计算网络风险时既考虑节点又考虑链路的后果很重要。

在第 9 章中，我们说明了网络拓扑结构对网络的稳定性是至关重要的。拓扑结构取决于节点之间的连接，这是映射函数 f 的一个特性。稳定的网络可以因为删除一条链路更改了网络的拓扑结构而变得不稳定。这里又会产生一个问题，“是否可以通过攻击一条链路，从而将稳定网络变成不稳定的网络？”我们将详细探讨这个问题。

11.2 关键节点分析

关键节点分析确定了在整体运行的网络中哪一个网络节点是最为关键的。我们知道，最重要的节点是那些要么有许多链路要么有大的目标价值从而导致重大的后果的节点。我们还知道，删除拥有更多链路的节点会造成更大的破坏，除非是在节点的价值有极大区别的异构网络中。我们所不知道的是在有限的预算（资源）及在具有各种变化的目标价值和保护每个节点与链路的费用不同的异构网络中，如何最好地保护管理节点。

关键节点分析是一种同时考虑节点/链路目标价值和网络拓扑结构的静态技术。强化关键节点的想法最简单直观，因此就从这个想法开始。假设将网络风险定义成拓扑结构、节点/链路后果和强化每个节点/链路的费用。以预算的形式下给定有限的资源，如何最佳地为每个节点/链路分配资源才能使得网络风险最小化？这是关键节点的资源分配问题：

B：固定预算

DA_i：节点/链路 $i = 1, 2, \cdots, (n+m)$ 的投资量

$v_i(DA)$：组件 i 的脆弱性，作为投资的函数

$\max DA_i$：使 $v_i(DA)$ 减少到最低限度的投资量

c_i：成功攻击节点/链路 i 的后果

$$R = \sum_{i=1}^{n+m} r_i = \sum_{i=1}^{n+m} v_i(DA)c_i$$

约束：$\sum_{i=1}^{n+m} DA_i \leqslant B$

下列策略为网络防御者导出最优资源的配置，重点是找出最关键的节点并按序分配最大部分的预算，直到耗尽预算为止：

1. 定义一个表示异构节点和链路（因为它们有不同的目标价值和强化成本）、网络连通性（以网络度序列分布形式）的目标函数 Φ。最小化目标函数也就是最小化网络风险。

2. 在脆弱性和成本之间定义一个合理关系，我们为资产（节点或链路）分配的资源越多，脆弱性就越低。因此，通过“买断”脆弱性可以减少风险。回顾脆弱性等于成功攻击的概率，风险为乘积 TVC。买断 V 也可以减少风险。

3. 通过最大限度地减少目标函数，在约束内求解节点/链路分配 DA_i。因此，我们所要的分配是一个优化的副产品。

结果证明，目标函数可以有多种定义方式。或许最简单的目标函数也与网络风险相关，将目标函数 Φ 定义为网络中每个节点－链路－节点子图贡献的风险总和，如图 11-1 所示。因此，每个网络可被视为一个杠铃子图系统，每一个杠铃都对整体网络贡献一部分的风险。这个定义的含义容易理解，并且杠铃模型保留了我们所寻求的度序列性质。

简单的杠铃模型还可以方便地集成链路和节点的异构目标值。每个杠铃包含两个节点和一条链路。两个节点中的任意一个和连接链路有一个关联的后果为 c_i，脆弱性消除或减少费用为 DA_i，故障概率为 $v_i(DA_i)$。给定一个将投资 DA_i 与脆弱性关联起来的公式 $v_i(DA_i)$ 和目标函数 Φ，就可以找到优化分配矢量 DA。

11.2.1 杠铃模型

图 11-1a 显示了如何将一个任意的网络分解成多个节点－链路－节点杠铃。在一个 m 条链路的网络中会有 m 个杠铃，每一个杠铃都对应一条链路。杠铃的重叠部分是有目的的，我们要用节点和链路的单独风险以及网络的连通性来定义目标函数 Φ。重叠杠铃可以这么做是因为一个度为 g 的节点一共计数 g 次——每次重叠杠铃就计数一次。

目标函数 Φ 定义为在预算约束 $\sum_{i=1}^{n+m} DA_i \leqslant B$ 条件下所有杠铃风险的总和。r_j 为杠铃 j 的相关风险，不受约束的目标函数为 $\Phi = \sum_{j=1}^{m} r_j$，其中 m 为链路数量。

杠铃风险 r_j 是从图 11-1b 中的故障树中得到的。一个杠铃很容易受到删除一个节点、链路或节点和链路组合的攻击。在以 OR 逻辑建模表示一个、两个或三个组件全部发生故障的相关概率 V_L、V_k 和 V_R 的故障树中可以找到所有可能的组合。将故障树扩展到一个事件树显示了所有可能出故障的组合以及每个组合的风险（参见图 11-2）。无约束目标函数只是简单的单个事件风险的总和，以便获取杠铃风险 r_j。

故障树代表了系统可能失效的所有方式。它们

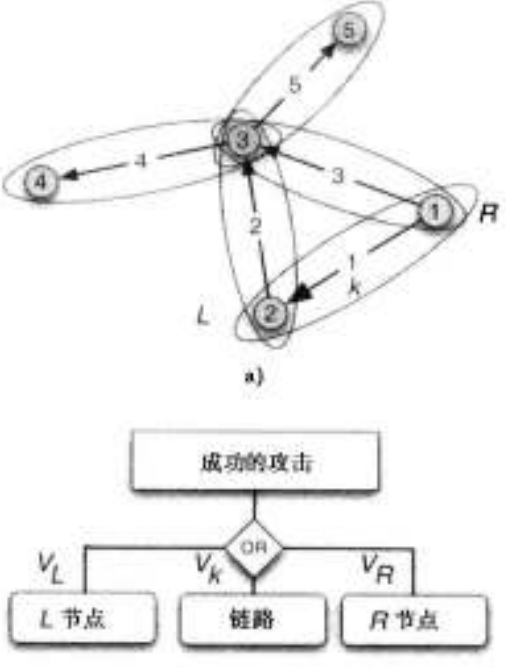


图 11-1 作为杠铃子图系统的网络：a）网络杠铃——根据网络的连通性，每个节点－链路－节点杠铃与其他相互重叠；b）杠铃故障模式的故障树模型

的深度可以有许多层次，可包含如 AND、OR 和 NOT 逻辑门。一个 OR 逻辑门意味着如果组件 A 或 B 或两者都失败，便会沿着故障树传播。AND 逻辑门是指只有当 A 和 B 都故障时才会传播故障。组件显示成故障树中的叶子，既可以失败也可以完好，取决于它们的脆弱性。我们将潜在故障标记成故障概率 V，并使用布尔逻辑和概率理论来计算一个或多个组件故障将传播到树的顶部从而导致系统失败的可能性。

考虑图 11-1a 中第 k 个杠铃的故障树，如图 11-1b 所示。在该树的顶端是攻击者的目标——导致系统失效。目标框下面的 OR 菱形逻辑框说明考虑了失败的所有可能的组合。如果 L 节点以概率 V_L 被成功攻击，链路 k 以概率 V_k 被成功攻击，节点 R 以概率 V_R 被成功攻击，或者节点与链路的任意组合被成功攻击，杠铃被成功攻击。在这种情况下有三个组件可能会失效，因此要考虑 $2^3=8$ 种可能事件。

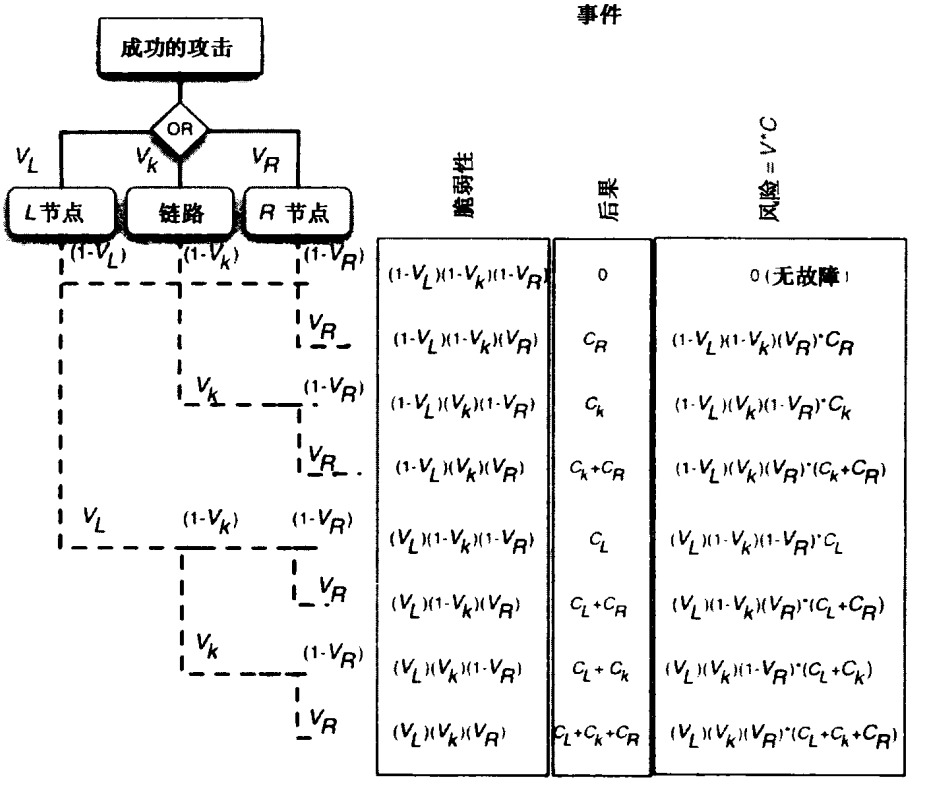


图 11-2 图 11-1 的杠铃 k 的事件树。对于一个具有 p 个组件的故障树，必须考虑 2^p 种事件及其组合。在这种情况下，$p=3$，所以必须考虑 8 种事件

此外，C_L、C_k 和 C_R 分别是节点 L、链路 k 和节点 R 遭到成功的攻击所带来的后果。我们将基本的 PRA 风险方程扩展到相应的事件树以便可以应用到故障树（参见图 11-2）。一个事件树简单枚举了由故障树 OR 逻辑产生的所有可能的事件。因为故障树有三个可能发生故障的组件，所以有 8 种事件。每种组合可能以沿着从故障树路径出现的事件组合的概率的概率乘积计算，如图 11-2 中的虚线所示。例如，考虑组合：

L 节点：失败的概率 V_L

链路 k：没有失败的概率 $(1-V_k)$

R 节点：失败的概率 V_R

这些事件组合的概率就是沿虚线导致对应事件的概率的乘积：

$$\text{Prob}(L\text{ 失败},k\text{ 成功},R\text{ 失败}) = V_L(1-V_k)V_R$$

这个概率对应于故障树到该组合事件的脆弱性，如图11-2所示的脆弱性列。

同样，两个节点L和R故障的后果就是后果总和：$C_L + C_R$。这是显示在图11-2中的“后果”列。最后，我们将基本的PRA风险方程应用到这个组合事件以便得到风险贡献为：

$$\text{Risk}(L\ 失败, k\ 成功, R\ 失败) = V_L(1 - V_k)V_R(C_L + C_R)$$

由所有可能故障组合导致的总风险贡献等于这种杠铃对整个网络风险的贡献r_k：

$$\begin{aligned} r_k = {} & (1 - V_L)(1 - V_k)(V_R)C_R + (1 - V_L)(V_k)(1 - V_R)C_k + (1 - V_L)(V_k) \\ & \times (V_R)(C_k + C_R) + (V_L)(1 - V_k)(1 - V_R)C_L + (V_L)(1 - V_k)(V_R) \\ & \times (C_L + C_R) + (V_L)(V_k)(1 - V_R)(C_L + C_k) + (V_L)(V_k)(V_R)(C_L + C_k + C_R) \end{aligned}$$

后果分解因式并合并后，我们得到

$$r_k = V_R C_R + V_k C_k + V_L C_L$$

更一般地，$r_k = \sum_{j \in k} V_j C_j$，$j$的范围是杠铃$k$的两个节点和一条链路。

现在，对扩展风险推导的第二步，将目标函数Φ定义为所有杠铃风险的总和，这意味着我们计算每个节点g_i次，因为每个节点是g_i个杠铃的成员：$\Phi = \sum_{k=1}^{m} r_k = \sum_{k=1}^{m} \sum_{j \in k} V_j C_j = \sum_{k=1}^{n+m} g_k V_k C_k$，其中对于$k > n$，$g_i = 1$。换句话说，对于链路则$g = 1$；对于节点则$g$ = 度。这个方程的含义不言自喻，Φ是节点加权风险加上其链路风险的总和。节点要比链路对Φ贡献更多，因为删除一个节点不仅影响节点本身，而且影响到与之相连的所有链路。因此，我们在集成任意网络拓扑的同时必须为网络保留风险PRA的意义。这个目标函数将网络拓扑与风险关联起来。

例如，考虑图11-1a中的网络，具有在表11-1中给出的属性。假设没有受到保护，每个节点和链路都100%地易于被清除。因此，$V_k = 1$。节点1有两条链路，那么$g_1 = 2$，其删除的后果是10个单位。因此它的贡献为2（1）（10）=20。有5条链路，所有都具有同样的风险，所以链路贡献的风险为5（1）（1）（10）=50。将所有贡献加起来就产生一个170个单位的目标函数值。同一网络的PRA风险是110个单位，之所以较小，是因为PRA风险忽略了节点度。

表11-1 图11-1a的网络的属性

节点/链路	g	C	max DA	$r.gC$
节点1	2	10	15	20
节点2	2	20	5	40
节点3	4	10	10	40
节点4	1	5	5	5
节点5	1	15	1	15
链路1	1	10	1	10
链路2	1	10	1	10
链路3	1	10	1	10
链路4	1	10	1	10
链路5	1	10	1	10
总共	—	110	41	170

11.2.2 网络风险最小化

从表11-1中，通过查看r列下的值，我们可以确定对目标函数最大贡献的组件在图11-1a中例子网络中的位置。节点2和节点3每个贡献了总的40/170 = 23.5%，占了最大的部分。节点2的贡献是由于其高的后果20，节点3的贡献主要是由于其高连通性。

直观上，这些高值的节点应该接收到最多的投资，因为它们对Φ贡献了大部分。但是必须

考虑第三个因素：脆弱性成本的减少，max DA。最关键的节点可能会有高价值目标，但强化它们的成本也会很高。因此，对于这些高价值的节点的资源利用率偏低——或许很低。防御优化配置的关键是平衡资源利用以便抵消后果。

风险减少是降低后果、脆弱性或同时降低两者的过程。资源优化是从有限的资源中获取最大化的过程。在这种情况下，将有限的资源更多地应用到节点2上而不是节点3上，因为减少节点2的脆弱性（$\max DA_2 = 5$）要比减少节点3（$\max DA_3 = 10$）的脆弱性便宜。这是关键节点分析底层的概念——一种试图减少风险同时最大限度地提高资源利用率的方法。

线性成本模型

很显然，风险减少最简单的策略是随机或均匀地将资源分配到所有节点和链路上，也就是说，给定一个预算 B，在全部资产 $(n+m)$ 上平均分配 B。因此，$DA_i = B/(n+m)$，风险就会相应地降低：

$$R = \sum_{k=1}^{n+m} V_k C_k = \sum_{k=1}^{n+m} \left(1 - \frac{B}{(n+m)}\right) C_k$$

$$R_{\text{norm}} = \frac{\sum_{k=1}^{n+m} \left(1 - \frac{B}{(n+m)}\right) C_k}{\sum_{k=1}^{n+m} C_k}$$

为一个资产分配比 max DA_i 还大的资源是没有意义的，所以脆弱性不能被减少到低于零。例如，随机/均匀地将 $B = 20$ 分配到图11-1的网络，由表11-1给定属性，将PRA风险从110个单位减少到31.67单位如下：

$$DA_i = \frac{B}{(n+m)} = \frac{20}{10} = 2$$

$$v_i = \max\left\{\left(1 - \frac{2}{\max DA_i}\right), 0\right\}$$

$$R_{\text{norm}} = \frac{\sum_{i=1}^{n+m} v_i c_i}{\sum_{i=1}^{n+m} c_i} = \frac{\sum_{i=1}^{n+m} \max\left\{\left(1 - \frac{2}{\max DA_i}\right), 0\right\} c_i}{110} = \frac{31.67}{110} = 28.8\%$$

在这种情况下，我们假设随着资源分配的增加，脆弱性线性下降。我们还假设均匀分配到所有节点和链路，投资 B 回报率最高——并会导致风险显著地减少。但是防御者能够做得更好吗？假设节点/链路安全和节点/链路保护上投资规模大小之间存在线性关系，是否有可能将更多资源分配到关键节点/链路上并且进一步降低风险？

假设脆弱性和资源分配之间呈如下的线性关系。设脆弱性下降与保护每个节点/链路的投资成比例。此外，为关键节点/链路分配更多的预算 B，而在较不关键的节点/链路中分配较小的预算。实际上，我们可以不为某些节点/链路分配资源，而将大量资源分配到其他节点/链路上。我们将这种模型称为降低风险的线性成本模型。

为保护资产 k 分配越多的资金 DA_k，该资产就越不脆弱——最高达到最大投资 $\max DA_k$，如下所示：

$$V_k(DA_k) = 1 - \alpha_k DA_k; 0 \leqslant DA_k \leqslant \max DA_k$$

其中：

- DA_k = 强化节点/链路 k 而分配的资源；
- $\max DA_k$ = 完全消除脆弱性而需要的资源分配量；
- α_k = 直线斜率，从而使 $0 = 1 - \alpha_k \max DA_k$。

斜率是由100%的强化成本也就是 $\max DA_k$ 决定。当 $DA_k = \max DA_k$ 时，脆弱性趋向于零，所以 $\alpha_k = 1/\max DA_k$。这产生了简单的降低风险线性成本模型：

$$\Phi = \sum_{k=1}^{n+m} g_k V_k C_k = \sum_{k=1}^{n+m} g_k C_k \max\left\{\left(1 - \frac{DA_k}{\max DA_k}\right), 0\right\}$$

资源受防御者预算 B_D 的限制：$\sum_{k=1}^{n+m} DA_k \leqslant B_D$；$DA_k \geqslant 0$。

在线性模型的假设下，什么样的节点/链路资源分配模型能最大限度地减少风险？具体来讲，在如下预算约束下，找到 DA_k，$k = 1, 2, \cdots, n, (n+1), \cdots, (n+m)$，使得 Φ 最小化：

$$\min\{\Phi(DA)\} = \min \sum_{k=1}^{n+m} g_k C_k \max\left\{\left(1 - \frac{DA_k}{\max DA_k}\right), 0\right\}$$

$$\text{条件为} \sum_{k=1}^{n+m} DA_k \leqslant B_D; DA_k \geqslant 0$$

其中 DA = 我们寻找的向量分配结果；

g = 节点度序列；1 用于链路；

C = 后果矢量；

$\max DA$ = 将脆弱性降低到零的成本减少矢量；

B_D = 防御者的预算。

Lewis 和 Al-Mannai 证明，节点和链路的资源最佳分配模式使用简单的排序。分配是通过排序节点和链路进行的，根据 $g_k C_k / \max DA_k$ 乘积从最高到最低值，然后通过分配 DA_k 单位到最高排序资产直到预算 B_D 耗尽为止，以便完全消除脆弱性。所有剩余资产获得零分配。

直观上来讲，这种分配最大化了成本－效益比率，以资源利用率的方式在最有效率的资产上分配更多资源，而在较小效率的资产上分配较少的资源。具体来说，高后果并且低成本的资产会获得最大的保护，而高成本并且低后果的资产得到最少的保护。

为了计算给每个节点和链路的实际分配，节点和链路要根据自己的加权后果值进行排序，其中 j 根据乘积 $g_k C_k / \max DA_k$ 按升序列举了资产：

$$g_{j1} C_{j1} \max DA_{j1} \geqslant g_{j2} C_{j2} \max DA_{j2} \geqslant \cdots \geqslant g_{j\cdots} C_{j\cdots} \max DA_{j\cdots}$$

其次，将 $\max DA_{j1}$ 分配给最高的，$\max DA_{j2}$ 给次高的，依此类推，直到剩余的预算比 $\max DA_k$ 少为止。剩余预算 $\sigma < \max DA_k$ 分配给第 k 个排名的资产，而分配给所有剩余资产的资源为零。这样，链路/节点以最有效的方式使用资源，给予最高优先级和可能最高的资源。

排序分配策略是最优的，因为首先它有效地减少了最高风险/回报节点/链路的贡献，直到预算耗尽为止。因此，排序分配维护了按加权后果建立的排序属性：

$$g_{j1} C_{j1} \frac{\max DA_{j1}}{\max DA_{j1}} \geqslant g_{j2} C_{j2} \frac{\max DA_{j2}}{\max DA_{j2}} \geqslant \cdots \geqslant g_k C_k \frac{\sigma}{\max DA_k} \geqslant 0 \geqslant 0 \geqslant \cdots \geqslant 0$$

$$g_{j1} C_{j1} \geqslant g_{j2} C_{j2} \geqslant \cdots \geqslant g_k C_k \frac{\sigma}{\max DA_k} \geqslant 0 \geqslant 0 \geqslant \cdots \geqslant 0; \frac{\sigma}{\max DA_k} < 1$$

因此，最佳线性防御分配为：

$$DA_{j1} = \max DA_{j1}$$

$$DA_{j2} = \max DA_{j2}$$

$$\vdots$$

$$DA_k = \sigma = \frac{\text{剩余 } B}{\max DA_k}$$

$$DA_i = 0; k < i \leqslant n+m$$

根据分配和消除脆弱性的成本 $\max DA_k$ 之间的线性关系，这种分配减少了脆弱性：

$$V_k(DA_k) = 1 - \alpha_k DA_k = \left(1 - \frac{DA_k}{\max DA_k}\right)$$

例如，给定一个20个单位的预算及如图11-1a和表11-1所示的简单网络，PRA风险从110减少到16.0个单位（14.5%），并且网络风险 Φ 线性分配由170个减少到29个（17.1%）——与随机/均匀分配相比结果提高了50%。表11-2中的数据总结了这种计算。更为重要的是，线性分配服从由 $g_iC_i/\max DA_i$ 乘积建立的排序，查看标记为 $gC/\max DA$ 的列和图11-3。这种属性决定了分配优先权——从最高到最低：节点5，链路1~5，节点2，节点3，节点1和节点4。事实上，在分配策略中已经观察到这个属性，而与分配和减少脆弱性的关系是线性还是指数或幂律分布无关！这在节点和链路间建立了层次化结构，网络上最关键的节点/链路是那些拥有最高的 $gC/\max DA$ 值的节点/链路。

请注意，最佳分配模式可能不是唯一的。当各节点和链路间具有排序联系时，在对等之间的任意分配可产生相同的最低风险。尤其是当没有足够的预算 $\max DA$ 分配到同等级别的所有资产中时，这特别正确。在这种情况下，可能产生多个最优解。

表11-2 对于 B_D 的线性成本模型结果

节点/链路	g	C	gC	$\max DA$	$gC/\max DA$	DA	V（%）	Φ
节点1	2	10	20	15	1.33	0.0	100	20.0
节点2	2	20	40	5	8.00	5.0	0	0.0
节点3	4	10	40	10	4.00	9.0	10	4.0
节点4	1	5	5	5	1.00	0.0	10	5.0
节点5	1	15	15	1	15.00	1.0	0	0.0
链路1	1	10	10	1	10.00	1.0	0	0.0
链路2	1	10	10	1	10.00	1.0	0	0.0
链路3	1	10	10	1	10.00	1.0	0	0.0
链路4	1	10	10	1	10.00	1.0	0	0.0
链路5	1	10	10	1	10.00	1.0	0	0.0
预算=20	—	110	170	41	—	20	总计 R=	29.0

进一步来讲，给定假设随着投资增加保护呈线性增加，优化分配的确是最好的可能分配。任何其他分配并不违反线性假设，其限制导致了一个同等或后果更高的网络风险——但不一定是PRA风险，因为 Φ 为目标函数。

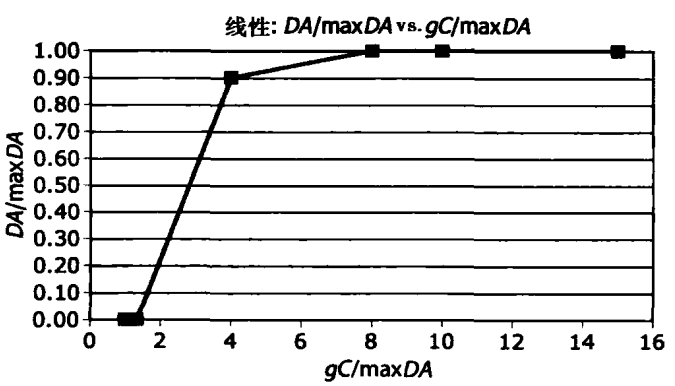


图11-3 线性分配策略相当于节点和链路按照 $gC/\max DA$ 排序。在这个例子中最重要的资产是节点5，因为 $gC/\max DA$ =15，这是对所有节点和链路中的最大值

例如，在节点4和5之间交换一个单位的资源分配会将网络风险由29个单位增加至48个单位！其证明留给读者作为练习。

11.2.3 指数成本模型

可以令人信服地认为，线性成本模型是不切实际的。在实际中，节点的安全性可能会以花费10%的预算却增加50%，但另外20%的提高则必须分配两倍的同样的预算。例如，在10 000美元的基础上增加更多照明费用，建筑物的安全可能增加了50%，虽然建设20 000元围栏也可提高建筑物的安全，但却只能增加10%的安全。换言之，脆弱性减少会受到收益递减的影响。正因为此，指数成本模型可能更为可取。

除了分配和脆弱性减少之间的关系之外，指数模型与线性模型完全一样。此外，分配策略是相同的，最高级别的节点/链路根据排序性质，$gC/\max DA$，会比低排名收到更多的资源。事实上，只要逐渐减少的资源受到逐渐减少的收益的影响，分配就服从我们前面已经看到过的熟悉的S形曲线。

指数成本模型在两个重要方面不同于线性模型：（1）实际分配DA是不同的；（2）网络风险一般会更高，因为需要无限地投资才能完全消除脆弱性。脆弱性减少的一个简单指数函数为：

$$v_i(DA_i) = e^{-\alpha_i DA_i};0 \leqslant v_i(DA_i) \leqslant 1$$

显然，这个函数值会渐近下降到零，但是只有在将无限大的资源分配给该资产时才能到达零。与完全消除了最关键节点和链路的脆弱性的线性策略不同，指数成本分配从来没有完全消除脆弱性。

当目标函数Φ最小化时，分配到节点和链路的预算B是最优的，使用下面的预算约束：

$$\Phi = \sum_{i=1}^{n+m} g_i v_i c_i = \sum_{i=1}^{n+m} g_i e^{-\alpha_i DA_i} c_i - \lambda_1 \left[\sum_{i=1}^{n+m} DA_i - B \right]$$

读者可以证明，优化分配向量DA可以由下列公式获得：

$$DA_i = \frac{\log(\alpha_i g_i C_i) - \log(\lambda_i)}{\alpha_i}$$

$$\log(\lambda_i) = \frac{\sum_{i=1}^{n+m} \frac{\log(\alpha_i g_i C_i)}{\alpha_i} - B}{\sum_{i=1}^{n+m} \frac{1}{\alpha_i}}$$

将该模型应用到图11-1和表11-1所示的简单网络中，使用20个预算单位，结果降低网络风险$\Phi = 37.9$个单位（37.9/170 = 22.3%），这比线性分配更高，因为指数模型从来没有将脆弱性减少到零。但是，比起随机/均匀分配，该分配产生更低的风险，因为指数分配服从$gC/\max DA$排序。解矢量DA的元素可以通过插入到上述方程中而找到，这留给读者作为练习。

11.2.4 攻击者－防御者模型

Al-Mannai和Lewis从两个方向扩展了线性成本模型（Al-Mannai，2007）：（1）指数成本模型替代了线性成本模型；（2）Al-Mannai和Lewis以攻击者的形式增加了一个对手。此配置创建了两个团队竞争游戏，其中防御者试图减少风险，而攻击者试图增加风险[⊖]。用数学术语表述：这是一个两方参与的游戏，攻击者最大化目标函数，而防卫者最小化目标函数。

在上一节中描述的指数成本模型足以作为防御者分配模型。同样，攻击者的分配策略是建模成一个收益递减的指数函数。由于攻击者的脆弱性模型镜像反映了防御者的指数模型，我们说，攻击者－防御者的问题是对称的。从军事上来说，对称冲突等效于武力对战，其中每个竞争

⊖ 这与经济学中的Stackelberg领导策略相类似。

者与其他人一样都有同样的优势和缺点。对称的武力对战，通过灵活性及运用超级资产取得胜利。攻击者－防御者网络的问题并没有什么不同。

攻击者也必须在预算 B_A 之内，和防御者受到预算 B_D 的限制一样。攻击者的目标是分配 AA_k 美元到节点/链路 k 上以增加攻击成功的概率，而防御者分配 DA_k 以降低脆弱性。双方都必须在各自预算约束下分配其资源。我们考虑两种情况：（1）对手知道彼此的分配策略；（2）各对手的分配策略未知。事实证明，第一种情况下攻击者的分配策略将与第二种情况有着根本的不同。

图 11-4 说明了攻击者和防御者的指数成本模型。从攻击者的角度来看，少量的投资会使脆弱性快速增加，随着投资的增大会使脆弱性的增加逐渐减少。从理论上讲，需要无限的投资才会使脆弱性为 100%。相反，相应的投资为 maxAA 美元时脆弱性增加到 maxAV%，如图中点线所示。这两个数字确定了攻击者的脆弱性曲线 g 的形状：

$$\gamma_i = \frac{-\ln(1 - \max AV_i)}{\max AA_i}; 0 \leqslant \max AV_i \leqslant 1$$

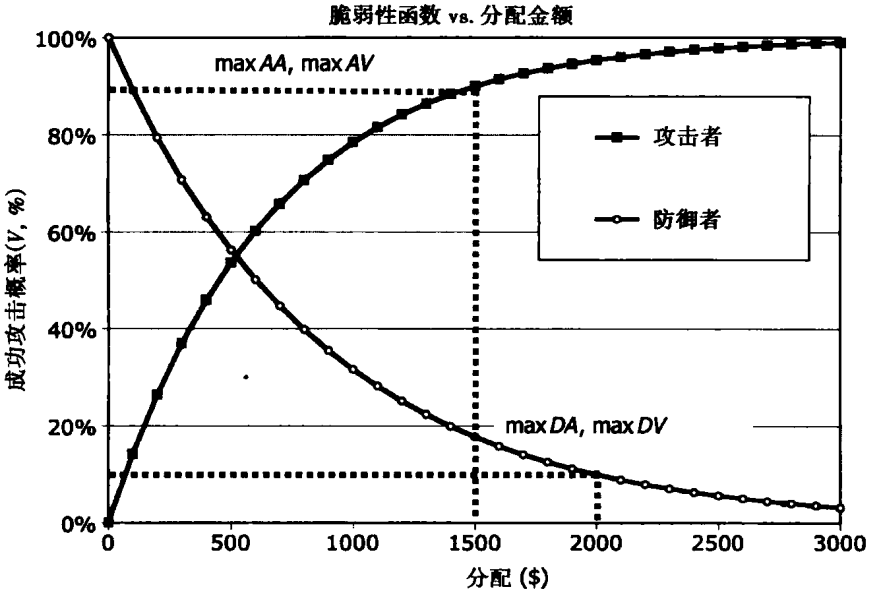


图 11-4 指数成本模型。随着投资的增加，攻击成功的概率也在增加；而防御者随着投资的增加其脆弱性降低

从防御者的角度来看，一开始少量的投资会导致脆弱性快速下降，然后随着投资增加而脆弱性的减少趋于平缓。当脆弱性渐近于零时，则需要无限大的投资才能使它等于零。相反，防御者可以投资 maxDA 美元使得脆弱性为 maxDV%。这两个数字确定了防御者的脆弱性曲线 α 的形状：

$$\alpha_i = \frac{-\ln(\max DV_i)}{\max DA_i}; 0 \leqslant \max DV_i \leqslant 1$$

攻击者和防御者都必须遵守收益递减的规则。两个指数代表了竞争双方的脆弱性。攻击者试图提高脆弱性，而防御者试图降低它。在每一种情况下，随着竞争对手花费有限的预算，脆弱性会上升或减少。因此，这些指数将每一节点和链路的投资与成功攻击的概率联系起来。效果上，攻击者的脆弱性 $V(AA)$ 以指数形式与攻击者的分配 AA 相当，而防御者的脆弱性 $V(DA)$ 以指数形式与防御者分配 DA 相当。这场冲突主要集中在每个竞争对手的预算量以及各自战略性分配资源的能力方面。

更正式地，设防御者的指数成本模型是一个简单的递减指数，速率常数由参数 maxDA 和

maxDV确定，由图 11-4 中点线标示：

$$v_i(DA_i) = e^{-\alpha_i DA_i};0 \leqslant v_i(DA_i) \leqslant 1$$

其中

$$\alpha_i = \frac{-\ln(\max DV_i)}{\max DA_i};0 \leqslant \max DV_i \leqslant 1$$

其中，当将 maxDA_i 投资到节点/链路 i 时，maxDV_i 为相应减少的脆弱性，DV_i 是由防御者分配的保护节点/链路 i 的投资。

请注意，在没有分配时，即 $DA_i = 0$ 时，脆弱性为 100%。另一方面，它需要一个无限大的分配才能完全消除脆弱性。选择参数 α_i 以便当 $DA_i = \max DA_i$ 时脆弱性降低到 maxDV_i。通常情况下，maxDV_i = 10%，如图 11-4 所示。

除了攻击者应用资金 AA_i 而使脆弱性增加之外，上述同样的论点可用于攻击者。因此，指数函数倒置：

$$v_i(AA_i) = 1 - e^{-\gamma AA_i};\ 0 \leqslant v_i(AA_i) \leqslant 1$$

其中

$$\gamma_i = \frac{-\ln(1 - \max AV_i)}{\max AA_i};0 \leqslant \max AV_i \leqslant 1$$

其中 maxAV_i 是对应于投资 maxAA_i 的脆弱性，AA_i 是分配给节点/链路 i 的投资。

再次，选择速率参数 γ_i 以校准每次输入 maxAV 和 maxAA 的指数曲线。通常情况下，maxAV_i = 90%，如图 11-4 所示。例如，给定 maxAV_i = 90%，maxAA_i = 1500 美元，minDV_i = 10%，以及 maxDA_i = 2000 美元，速率参数 α_i 和 γ_i 为：

$$\alpha_i = \frac{-\ln(0.10)}{2000} = 0.00115$$

$$\gamma_i = \frac{-\ln(1 - 0.90)}{1500} = 0.00154$$

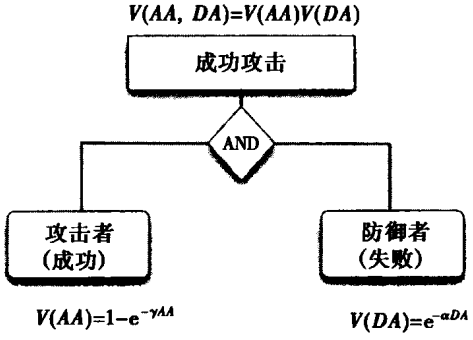


图 11-5 攻击者 - 防御者结合模型的故障树。攻击成功的概率等于成功攻击的概率与防御失败的概率乘积

注意攻击者和防御者的脆弱性是建模成 AND（与）门故障树的系统中事件的概率，两个独立的指数成本模型现在合并成单个模型。图 11-5 的故障树包含两个面对节点/链路的威胁：攻击者尝试成功攻击的概率为 $V(AA)$，防御者可能阻止攻击失败的概率为 $V(DA)$。请注意，攻击者在提高资源分配情况下攻击机会增加，而防御者随着资源分配的增加而降低失败。因此，一次成功攻击的联合概率是两个概率的乘积：

$$v_i(AA_i, DA_i) = v_i(AA_i)v_i(DA_i) = [1 - e^{-\gamma AA_i}]e^{-\alpha_i DA_i}$$

这个方程很重要，因为它将攻击者和防御者资源分配结合起来定义风险。例如，如果两个参

与者都没有分配什么，$AA_i = DA_i = 0$，合并后的脆弱性是100%。如果防御者没有分配并且$AA_i \geqslant 0$，那么脆弱性就取决于攻击者的分配，因此也就是攻击者的成功概率$V(AA)$。相反，如果攻击者没有分配，防御者分配资源就没有意义，因为$V(AA) = 0$。联合脆弱性等于防御者的脆弱性，$V(DA) = \exp(-\alpha DA)$。在这种情况下，防御者的最佳策略是将资源分配到其他地方。

这种分析需要考虑两种重要情况：（1）攻击者和防御者知道对方的资源分配情况并做相应分配调整；（2）攻击者和防御者对彼此战略都不知道，只是根据某些假设分配。我们依次对这些进行研究。

假设指数级的脆弱性减少成本方程，并假定攻击者和防御者知道对方的策略。攻击者试图最大化风险而防御者试图最大限度地降低风险。此外，我们使用前面定义的目标函数Φ作为网络风险，因为我们首先要保护更高度的节点而不是低度的节点。并且认为Φ考虑了网络的度序列，也考虑了后果和脆弱性减少成本。因此，攻击者的目的就是提高Φ，而防御者的目标就是减少Φ。

这是一个两方的博弈，在数学中表示成*攻击者－防御者优化问题*：

$$\max_{AA}\min_{DA}\{\Phi(AA,DA)\}$$

这里AA =攻击者的分配向量，DA =防御者的分配向量，并且

$$\Phi(AA,DA) = \sum_{k=1}^{n+m} g_k v_k(AA_k,DA_k)C_k = \sum_{k=1}^{n+m} g_k[1-e^{-\gamma_i AA_i}]e^{-\alpha_i DA_i}C_k$$

$$\sum_{i=1}^{n+m} AA_i \leqslant B_A; AA_i \geqslant 0$$

$$\sum_{i=1}^{n+m} DA_i \leqslant B_D; DA_i \geqslant 0$$

其中C_k代表了后果、目标值或破坏，B_A = 攻击者预算，B_D = 防御者预算。

此约束优化问题可以转化成无约束的优化问题，通过采用拉格朗日乘数λ_1和λ_2：

$$\max_{AA}\{\min_{DA}\{\Phi\}\}$$

$$\Phi = \left\{ \begin{array}{l} \sum_{k=1}^{n+m} g_k[1-e^{-\gamma_i AA_i}]e^{-\alpha_i DA_i}C_k \\ -\lambda_1\left[\sum_{i=1}^{n+m} DA_i - B_D\right] \\ -\lambda_2\left[\sum_{i=1}^{n+m} AA_i - B_A\right] \end{array} \right\} \quad \text{受限于 } AA_i \geqslant 0; DA_i \geqslant 0$$

现在的问题是要找到在预算受限情况下满足目标函数的DA和AA。解向量DA和AA是通过设置相对于AA、DA、λ_1和λ_2的导数等于零，并求由此产生的联立方程的解而获得的。这会导致出现巨大的计算量，但这个问题通过数学优化和算法迭代就很容易解决。我们遵循Al-Mannai和Lewis提出的混合分析和计算方式，导出了攻击者－防御者的最优分配策略。

Al-Mannai和Lewis为了得出AA和DA提出了*军备竞赛算法*，该算法使得防御者最小化风险而攻击者最大化风险。该算法很简单，除了当攻击者分配为零时会出现了一个奇点之外。在这种情况下，防御者分配也为零，因为一个明智的防御者不会在一个没有攻击者威胁的资产上浪费资源。

军备竞赛算法

1. 设置初始防御者分配向量DA为零。将攻击者预算均匀地分配到所有的节点和链路上：$AA_i = B_A/(n+m)$。计算所有节点和链路的α_i和γ_i。将所有节点/链路标记为可供分配：negativeAlloc = false。计算初始风险，假设故障概率为1：$R = \Sigma c_i$。

2. 重复下面的优化计算，直到联合风险几乎没有什么变化为止：
   a. 保持攻击者分配矢量 AA 为常量，并使用拉格朗日乘数法找到以矢量 AA 表示的最小防御者分配矢量 DA。如果任何分配为负，就设置分配为零，并标记节点/链路：negativeAlloc = true。在接下来的迭代中不再进一步考虑这些节点/链路。
   b. 为了最大化使用如上获得的防御者分配向量 DA，以及第二个拉格朗日乘数，找到相对前面已经计算出来的 DA 的攻击资源最优分配的 AA。如果任何分配为负值，设置分配为零，并标记节点/链路：negativeAlloc = true。在接下来的迭代中不再进一步考虑这些节点/链路。
   c. 使用从 AND（与）逻辑故障树中得到的失败联合概率的方程，计算一个新的风险，并与以前迭代的风险计算做比较。当变化可以忽略不计时就停止。

事实证明，该算法很快收敛为一个固定点，也就是说在该点处进一步改变 DA 和 AA 对联合风险的影响不大。像冷战时期的军备竞赛一样，参与者最终陷入僵局，若没有次优化的话进一步完善他们的分配策略是不可能的。这个固定点的偏差为攻击者和防御者产生一个次优策略。在这个固定点上，最好的资源分配存储在向量 AA 和 DA 中。

军备竞赛算法步骤 1 假定一个初始向量 AA 等于 $B_A/(n+m)$ 并允许防御者作出初步分配，使用攻击者分配向量以尽量减少风险。防御者试图最小化风险，所以通过求解向量 DA 以初始 AA 值的名义，我们找到防御者的最小化问题的临时解。对 DA_i 和 λ_1 求导就产生一组可解方程组：

$$\frac{\partial \Phi}{\partial DA_i} = 0 = -\alpha_i g_i C_i (1 - e^{-\lambda AA_i}) e^{-\alpha_i DA_i} - \lambda_1$$

$$\frac{\partial \Phi}{\partial \lambda_1} = 0 = -\sum_{i=1}^{n+m} DA_i + B_D$$

在第一个方程中参照 ln（λ_1）解出 DA_i，然后代入第二个方程参照向量 AA 解出向量 DA_i：

$$DA_i(AA_i) = \frac{\ln(\alpha_i g_i C_i) - \ln(\lambda_1) + \ln(1 - e^{-\gamma AA_i})}{\alpha_i}$$

$$\ln(\lambda_1) = \frac{\sum_{i=1}^{n+m} \left[\frac{\ln(\alpha_i g_i C_i) + \ln(1 - e^{-\gamma AA_i})}{\alpha_i} \right] - B_D}{\sum_{i=1}^{n+m} \frac{1}{\alpha_i}}; AA_i > 0$$

数值求解 ln（λ_1）很直观，因为右边的所有变量都是已知的。将 ln（λ_1）代入方程右边求解 DA_i 也是数值非凡的，因为向量 AA 从一开始就假定了。因此，向量 DA_i 是参照向量 AA 给出的，但 AA 是固定点的一个初步估计，而不是最终解。军备竞赛算法的下一步使用矢量 DA 求解“更好的 AA”。我们一直重复两个解，直到联合风险不可能再变化为止[⊖]。向量 AA 也是以相同方法计算，使用以前计算的向量 DA 和另一个拉格朗日乘数 λ_2：

$$AA_i(DA_i) = \frac{\ln(\gamma_i g_i C_i) - \ln(\lambda_2) - \alpha_i DA_i}{\gamma_i}$$

$$\ln(\lambda_2) = \frac{\sum_{i=1}^{n+m} \left[\frac{\ln(\gamma_i g_i C_i) - \alpha_i DA_i}{\gamma_i} \right] - B_A}{\sum_{i=1}^{n+m} \frac{1}{\gamma_i}}; AA_i > 0$$

⊖ 程序 NetworkAnalysis.jar 使用 Math.abs（SL-newSL）/newSL≤0.000001 作为停止标准，这里 SL 和 newSL 为随后的网络风险。

再次，$\ln(\lambda_2)$ 由将已知值代入右边方程来确定。然后，代入 $\ln(\lambda_2)$ 数值计算向量 AA，将 DA 代入右边方程计算 $AA_i(DA_i)$。对于一个大型网络，计算单调乏味，所以建议使用计算机程序，参见程序 NetworkAnalysis. jar。

细心研究分配方程可以发现当 $AA_i=0$ 时出现一个奇点，因为 $\ln(1-\exp(-\gamma_i AA_i))$ 是未定义的，当它们趋近于零时对数等于负无穷。这种奇点异常导致防御者分配失效。从逻辑上讲，当攻击者的分配为零时，我们可以将防御者分配成我们想要的任何值。但是，如果没有来自攻击者的任何威胁，防御者为资产分配为零也是有意义的，可将资源用到别的地方。这种奇异性由后面讨论的 Java 来执行处理。

$DA(AA)$ 和 $AA(DA)$ 的迭代直到风险收益率变化不大为止，所产生的固定点代表了攻击者和防御者在其预算约束下可以做的最好分配。如图 11-6 所示，对于图 11-1a 的简单 4 节点/链路的网络，通常快速收敛。锯齿形的收敛解是由于攻击者试图最大化风险，而防御者试图最大限度地降低风险。在每一步中，攻击者增加风险，接着是防御者降低风险。迟早，两条线会收敛到一个固定点或僵持的配置。

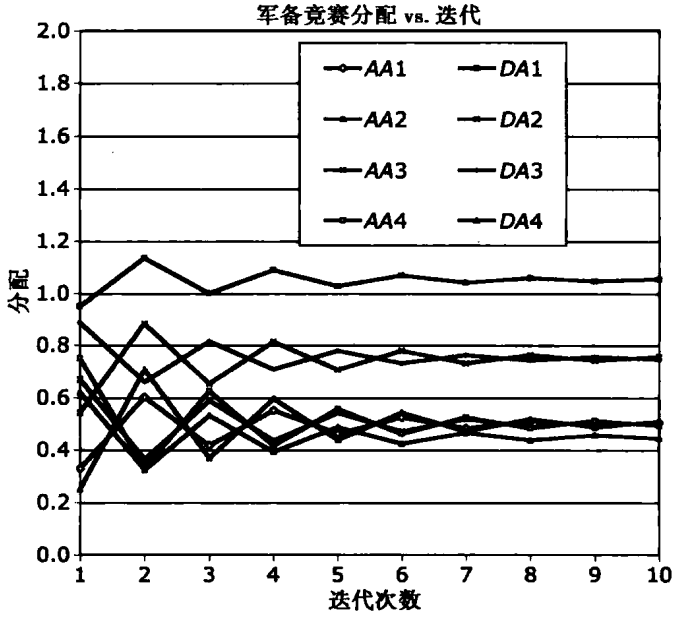


图 11-6 军备竞赛算法收敛速度快：攻击者和防御者的分配 AA 和 DA 与反复迭代军备竞赛算法

图 11-7 使用表 11-2 的数据比较了从图 11-1a 的简单网络取得的结果。像前面一样，分配遵循 $gC/\max$ Alloc 排列，标准化为分配/最大分配。曲线绘制仅对于相同数据的防御者线性和指数算法与军备竞赛算法对比。相同的预算、后果及脆弱性删除费用，使得比较公平。

这个例子说明，与之前线性算法一样，防御者分配与同一效益率 $gC/\max$ Alloc 相关。因此，线性和指数防御者策略按照最高加权效益/成本率 $gC/\max$ Alloc 将资源按序分配到资产。事实上，对于线性和指数策略曲线是彼此相近的。忽视所使用的脆弱性削减的成本方程式，效率分配遵循相同秩序规则。

接着，攻击者防御者的行为如图 11-7 中虚线所示。防御者紧随指数分配曲线，几乎逼近。但是，攻击者则刚好相反！攻击者往往会不对称分配资源——与防御者相反。换言之，防御者的低资源分配导致节点/链路的脆弱性，这会为攻击者提供机会。因此，攻击者会为这些未很好地受到保护的资产分配更多的资源，而减少对其他目标资源的分配。我们总结出最佳攻击者的策略为不对称——攻击较弱的节点/链路，而忽视受到良好保护的节点/链路。

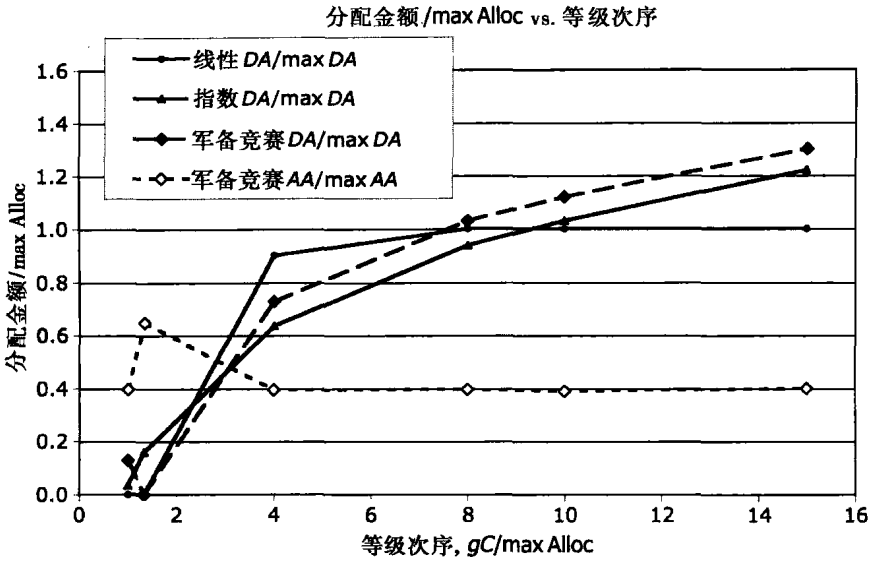


图 11-7 仅限于防御者线性策略和仅限于防御者指数策略曲线，与军备竞赛算法中攻击者和防御者的资源分配比较。数据是以等级次序按 g_iC_i/max Alloc$_i$ 升序排列。攻击者和防御者参数是相同的，包括预算：攻击者预算 = 防御者预算 = 20 个单位

最后需要解释的是图 11-7 中的曲线。当使用指数脆弱性减少函数而不是用线性函数时，风险会较高。这是正确的，因为指数函数跨越实线，从零到无穷大。根据指数模型需要无限的预算才能消除节点/链路的所有脆弱性。相反，线性模型在 max *DA* 分配处与零脆弱性相交；也就是说，如果线性模型假设有足够的和有限的预算，所有的脆弱性（因而风险）都可能会被删除。因此，相对于线性模型来讲，指数模型总是风险较高。但是，这只是一个人造模型而已，而不是网络的模型。

11.2.5 Java 军备竞赛方法

通过五种 Java 方法实现攻击者 - 防御者的分配算法：一种用于实现上面军备竞赛算法描述的步骤的根方法，两种方法用于计算 λ_1 和 λ_2，以及用于防御者和攻击者分配计算的各有一种方法。为处理攻击者 - 防御者竞争编写方法，也为独立的指数情况编写。当标记 competition 为 true（真）时，攻击者和防御者使用对方的分配来计算自己的分配。当为 false（假）时，双方都忽略对方的分配。在这种情况下，该分配与从指数脆弱性减少方程中获得的分配相同。第二个标志 network 用来指示在分配中使用的节点度。当 network 为 true 时，就使用度序列 g；否则将所有节点和链路 g 都设置为 1。

根方法 NW_doExponentialArmsRace( ) 无例外地解释了军备竞赛算法。最初，攻击者预算均匀地分布到所有节点/链路中，计算了指数参数 α 和 γ。负分配标志设置为 false，表明最初所有的分配是非负的。然后在防御者和攻击者分配之间迭代计算军备竞争算法。如果任何一方预算为零，分配遵循单方面指数策略：

```
public void NW_doExponentialArmsRace(boolean network, boolean competition){
    NW_Zero_Allocation();   //Reset Budget and Point allocation to zero
    for(int i = 0; i < nnodes; i++){
       nodes[i].AttackerAlloc = aBudget/(nnodes+nedges);
```

```
        nodes[i].alpha = -Math.log(nodes[i].DefenderFrac)/nodes[i].maxDA;
        nodes[i].gamma = -Math.log(1-nodes[i].AttackerFrac)/nodes[i].maxAA;
        nodes[i].negativeDefenderAlloc = false;
        nodes[i].negativeAttackerAlloc = false;
     }
     for(int i = 0; i < nedges; i++){
        edges[i].AttackerAlloc = aBudget/(nnodes+nedges);
        edges[i].alpha = -Math.log(edges[i].DefenderFrac)/edges[i].maxDA;
        edges[i].gamma = -Math.log(1-edges[i].AttackerFrac)/edges[i].maxAA;
        edges[i].negativeDefenderAlloc = false;
        edges[i].negativeAttackerAlloc = false;
     }
     SL = 0;
     newSL = 100;
     while(Math.abs((SL-newSL)/newSL) > .000001){
        if(aBudget > 0) doNetworkDefenderAllocation(network, competition);
          else doNetworkDefenderAllocation(network, false);
        if(dBudget > 0) doNetworkAttackerAllocation(network, competition);
          else doNetworkAttackerAllocation(network, false);
        SL = newSL;
        doExponentialV();  //Calculate combined vulnerability
            newSL = NW_calculateNetworkRisk();
         }
         doExponentialV();
         NW_calculateNetworkRisk();
   }//NW_doExponentialArmsRace
  private void doExponentialV(){
     double probA, probD;
     for(int i = 0; i < nnodes; i++){
          probA = 1-Math.exp(-nodes[i].gamma*nodes[i].AttackerAlloc);
          probD = Math.exp(-nodes[i].alpha*nodes[i].DefenderAlloc);
          nodes[i].ProbOfFailure = FaultTree(probA, probD);
     }
     for(int i = 0; i < nedges; i++){
        probA = 1-Math.exp(-edges[i].gamma*edges[i].AttackerAlloc);
        probD = Math.exp(-edges[i].alpha*edges[i].DefenderAlloc);
        edges[i].ProbOfFailure = FaultTree(probA, probD);
     }
  }
  private double FaultTree(double probA, double probD){
        if(probA < 0) probA = 0;
        if(probD < 0) probD = 0;
        if(probA > 1) probA = 1;
        if(probD > 1) probD = 1;
        if(dBudget == 0) return probA;
        if(aBudget == 0) return probD;
        return probA*probD;
      }
```

假定前面的分配，攻击者和防御者分配方法反复地计算和重计算最佳分配，直到所有的负分配都被删除为止。这就加强了问题公式要求的对 $AA > 0$ 和 $DA > 0$ 的约束。每次发现一个负分配，就会被删除，在更少的资产上重复分配。一旦删除，分配就会被设置为零，就不再改变。此

外，防御者方法避免了由攻击者零分配造成的奇点。如果攻击者节点/链路分配一个零，防御者也会这样做，并从进一步的考虑中删除节点/链路。因此，防御者策略与处理负分配策略是相同的。尽管这不是为防御者分配应用方程的直接结果，但在现实世界中是有意义的——既然没有威胁，那么为什么还要分配保护资源呢？

```
private void doNetworkDefenderAllocation(boolean network, boolean competition){
   boolean no_negatives = false;
   int degree;
   while(!no_negatives){
     no_negatives = true;
     double logLambda1 = NW_doNetworkLambda1(network, competition);
     for(int j = 0; j < nnodes; j++){
       if(!nodes [ j] .negativeDefenderAlloc){
     if(network) degree = nodes [ j] .g; else degree = 1;
     if(competition && aBudget > 0 && nodes [ j] .AttackerAlloc <= 0)
           nodes[j].DefenderAlloc = -1; //Remove it
     else {
         double VA = nodes [ j] .gamma*nodes [ j] .AttackerAlloc;
         if(!competition) VA = 0;
            else VA = Math.log(1-Math.exp(-VA));
         nodes[j].DefenderAlloc = (Math.log(nodes[j].alpha*degree*nodes[j].C)-
logLambda1+VA)/nodes[j].alpha;
      }
     }
     if(nodes [ j] .DefenderAlloc < 0){  //Remove negatives
        nodes[j].DefenderAlloc = 0;
        nodes[j].negativeDefenderAlloc = true;
        no_negatives = false;
     }
  }//nnodes
  for (int j = 0; j < nedges; j++){
     if(!edges [ j] .negativeDefenderAlloc) {
        if(competition && aBudget > 0 && edges [ j] .AttackerAlloc <= 0)
             edges[j].DefenderAlloc = -1;
        else {
          double VA = edges [ j] .gamma*edges [ j] .AttackerAlloc;
          if(!competition) VA = 0;
          else VA = Math.log(1-Math.exp(-VA));
        edges[j].DefenderAlloc = (Math.log(edges[j].alpha*edges[j].C)-
logLambda1+VA)/edges[j].alpha;
        }
     }
     if(edges [ j] .DefenderAlloc < 0){
        edges[j].DefenderAlloc = 0;
        edges[j].negativeDefenderAlloc = true;
        no_negatives = false;
     }
  }//nedges
 }
}//doNetworkDefenderAllocation
```

```
private void doNetworkAttackerAllocation(boolean network, boolean competition){
     boolean no_negatives = false;
     int degree;
     while(!no_negatives){
         no_negatives = true;
         double logLambda2 = NW_doNetworkLambda2(network, competition);
         for(int j = 0; j < nnodes; j++){
         if(!nodes [ j] .negativeAttackerAlloc) {
         double VD = nodes[ j] .alpha*nodes[ j] .DefenderAlloc;
         if(!competition) VD = 0;
         if(network) degree = nodes[ j] .g; else degree = 1;
         nodes[j].AttackerAlloc = (Math.log(nodes[j].gamma*nodes[j].C*degree)-
logLambda2-VD)/nodes[j].gamma
         if(nodes [ j] .AttackerAlloc < 0){
            nodes[j].AttackerAlloc = 0;
            nodes[j].negativeAttackerAlloc = true;
            no_negatives = false;
         }
        }
      }//nnodes
      for(int j = 0; j < nedges; j++){
        if(!edges[ j] .negativeAttackerAlloc) {
           double VD = edges[ j] .alpha*edges[ j] .DefenderAlloc;
           if(!competition) VD = 0;
           edges[j].AttackerAlloc = (Math.log(edges[j].gamma*edges[j].C)-
logLambda2-VD)/edges[j].gamma
           if (edges[ j] .AttackerAlloc < 0){
                 edges[j].AttackerAlloc = 0;
                 edges[j].negativeAttackerAlloc = true;
                 no_negatives = false;
           }
       }
   }//nedges
 }
}//doNetworkAttackerAllocation
```

参数 $\ln(\lambda_1)$ 和 $\ln(\lambda_2)$ 也以类似的方式计算，直到负分配已被删除没有必要迭代为止，因为它们已经被相应的分配方法删除。两种计算 λ 的方法检查相应的 negativeAlloc 标志，而忽略标记为负的节点/链路。该方法还必须避免由于零攻击者分配而造成的失效。这种情况可能吗？这留给读者作为练习。

```
public double NW_doNetworkLambda1(boolean network, boolean competition){
  double Lambda1 =0.0;
  double sumAlpha = 0.0;
  int degree;
  for(int i = 0; i < nnodes; i++) {
     if(!nodes[ i] .negativeDefenderAlloc)
     {
       if(network) degree = nodes[ i] .g; else degree = 1;
       double VA = nodes[ i] .gamma*nodes[ i] .AttackerAlloc;
       if(!competition || VA <= 0) VA = 0;
          else VA = Math.log(1-Math.exp(-VA));
```

```
        Lambda1 +=(Math.log(nodes[i].alpha*degree*nodes[i].C)+VA)/nodes[i].alpha;
        sumAlpha += (1/nodes[i].alpha);
      }
    }
        for (int i = 0; i < nedges; i++){
          if(!edges[ i] .negativeDefenderAlloc)
          {
            double VA = edges[ i] .gamma*edges[ i] .AttackerAlloc;
            if(!competition || VA <= 0) VA = 0;
               else VA = Math.log(1-Math.exp(-VA));
            Lambda1 += (Math.log(edges[i].alpha*edges[i].C)+VA)/edges[i].alpha;
            sumAlpha += (1/edges[i].alpha);
           }
          }// for nedges
          if(sumAlpha == 0) return 0;
          else Lambda1 = (Lambda1-dBudget)/sumAlpha;
          return Lambda1;
      }// NW_doNetworkLambda1

    public double NW_doNetworkLambda2(boolean network, boolean competition){
          double Lambda2 =0.0;
          double sumGamma = 0.0;
          int degree;
          for(int i = 0; i < nnodes; i++) {
              if(!nodes[ i] .negativeAttackerAlloc) {
                 double VD = nodes[ i] .alpha*nodes[ i] .DefenderAlloc;
                 if(!competition) VD = 0;
                 if(network) degree = nodes[ i] .g; else degree = 1;
                 Lambda2 += (Math.log(nodes[i].gamma*degree*nodes[i].C)-
                           VD)/nodes[i].gamma
                 sumGamma += (1/nodes[i].gamma);
               }
          }
          for (int i = 0; i < nedges; i++) {
             if(!edges[ i] .negativeAttackerAlloc) {
                double VD = edges[ i] .alpha*edges[ i] .DefenderAlloc;
                 if(!competition) VD = 0;
                  Lambda2 += (Math.log(edges[i].gamma*edges[i].C)-VD)/
                            edges[i].gamma
                 sumGamma += (1/edges[i].gamma);
             }
          }// for nedges
          if( sumGamma == 0 ) return 0;
          else Lambda2 = (Lambda2-aBudget)/sumGamma;
          return Lambda2;
    }// NW_doNetworkLambda2
```

11.3 博弈论的考虑

攻击者－防御者军备竞赛模型假定了两个网络对手使用同样的策略，并应用同样的指数成本模型。此外，它假定每次重新分配后，每个参与者都知道其他参与者的分配。这些假设在许多情况下可能有效，但如果攻击者和防御者不知道对方的分配策略会发生什么？具体来讲，当任何

一方都不知道另外一方的策略时，最佳的分配策略是什么？

我们转向博弈论来分析这个问题。在一个简单的两方博弈中，防御者采用了许多策略中的一种，通过假定它知道一些有关对手的分配策略；反之，攻击者也采用了许多策略中的一种，通过假定它知道有关防御者的策略的某些内容。在实际中，有关对手的假设可能是错误的。然而，一旦采用策略，就至少应用于1轮博弈中。

一个收益矩阵包含了来自每个参与者分配策略应用到每一方时产生的收益（利益）。在 $n \times n$ 收益矩阵中每个元素包含了规范化的网络风险——通过应用策略到每一个给定的网络并计算由此产生的网络风险。我们使用规范化的网络风险定义：

$$\Phi(AA,DA) = \sum_{k=1}^{n+m} g_k v_k(AA_k,DA_k)C_k$$

$$\Phi(AA,DA)_{\text{normalized}} = \frac{\Phi(AA,DA)}{\Phi(0,0)};\Phi(0,0) = \sum_{k=1}^{n+m} g_k C_k$$

例如，表11-3包含了用于两方博弈收益矩阵，由每个参与者从本章中描述的选择中选择一种策略。这个收益矩阵用于图11-1a的简单网络，并使用了表11-1和表11-2中的参数。它包括了运行军备竞赛算法的结果，为了进行比较，包括了带有度序列和不带度序列的目标函数。

出人意料的是，这两方的最佳战略都是线性分配，根据资产等级次序将资源分配到在最关键的节点/链路上。为什么呢？一种解释是，当使用指数函数时需要一个无限分配来完全消除脆弱性，但线性分配通过分配有限的量 max Alloc，可以将风险减少到零。线性和指数函数之间的差异导致了使用指数函数会有较高的风险估计。如表11-3中所示，指数－指数表项比起线性－线性分配差异相当显著。

博弈论模型与之前探索的攻击者－防御者模型有所不同，因为它假定每个参与者并不知道彼此的分配。攻击者－防御者模型将脆弱性结合到一个联合风险函数中，在这个博弈论公式中，我们假设了独立的成功概率。因此，攻击者和防御者分配资源，犹如其他对手并不存在。当知道攻击者和防御者都了解对方的分配时，攻击者可以利用这些信息的优势使得风险比起未知这些信息时更高。

当预算置为0时忽略相应的收益项，那么对于每个参与者的最佳策略是什么？攻击者和防御者可以选择随机、线性或指数分配，但任何一方都不知道对方的选择。例如，如果防御者因为线性分配策略包含最小的风险而选择了该策略，则攻击者可能选择指数分配策略，因为它包含了最大的风险。在这种情况下，收益是4.06%，收益单元位于防御者－线性和攻击者－指数的行和列的交叉点处。

在这个例子中，防御者也只能采用线性分配策略，因为不论攻击者采取什么样的策略，从防御角度来说风险都只会增加。同样，攻击者最差可以采用线性分配策略，因为与采用随机或指数防御策略相比，从攻击者角度来讲风险将会减少。如果不知道对手的分配策略，每个参与者从自身的角度必须选择看起来最好的策略。对于简单网络来讲，如果不考虑预算的话，正确吗？这留给读者作为练习。

表11-3 简单例子的收益矩阵

防御者（预算=20）	攻击者（预算=20）					
	无分配（%）	随机（%）	线性（%）	指数（%）	军备竞赛（$g \geq 1$）（%）	军备竞赛（$g=1$）（%）
无分配	100.0	55.10	82.90	77.70	77.70	74.80
随机	44.9	11.50	31.10	30.23	—	—
线性	17.5	3.21	2.11	4.06	—	—
指数	22.3	7.60	9.65	12.26	—	—
军备竞赛（$g \geq 1$）	22.30	—	—	—	15.95	—
军备竞赛（$g=1$）	25.10	—	—	—	—	17.25

11.4 一般的攻击者 - 防御者网络风险问题

网络攻击者 - 防御者问题是不对称的，也就是说，因为攻击者具有较少的限制，它比防御者有（不公平的）优势。一般而言，攻击者可以选择地点、时间和威胁，而防御者除了防御一切、在所有的时间、对所有的威胁之外具有很少的选择。这种攻击者策略中的不对称来自以下的不平衡。

攻击者问题。攻击者不受攻击内容、攻击时间以及攻击手段的约束，也就是说攻击者可以使用任何武器在任何时间攻击网络的任意部分。然而，像防御者一样，攻击者受到费用成本的限制。攻击不是免费的。

防御者问题。防御者受防御内容、防御时间以及防御方式的限制，也就是说，防御者不能在所有时间不受限制地防护网络中的一切。防御者受维护节点和链路的费用的约束。安全不是免费的。

因为这种不对称，攻击者 - 防御者问题的解是非平凡的。一般来说，防御者是无法抵御一切威胁的，而攻击者是资源有限的，承担不起使用任何力量攻击所有部分的费用。

这就导致了博弈论方法，即防御者只保护最重要的部件，而攻击者尝试最大限度地破坏。在博弈论中，防御者试图最大限度地最小化风险，而攻击者试图最大化风险。更规范地，我们定义了一般的攻击者 - 防御者问题如下：

$$\Phi = \left\{ \begin{array}{l} \sum_{k=1}^{n+m} g_k V_k(AA_k, DA_k) C_k \\ -\lambda_D [\sum_{i=1}^{n+m} DA_i - B_D] \\ -\lambda_A [\sum_{i=1}^{n+m} AA_i - B_A] \end{array} \right\}; AA_k \geqslant 0; DA_k \geqslant 0$$

解

$$\frac{\partial \Phi}{\partial DA_i} = 0; \frac{\partial \Phi}{\partial \lambda_D} = 0$$

$$\frac{\partial \Phi}{\partial AA_i} = 0; \frac{\partial \Phi}{\partial \lambda_A} = 0$$

这种风险问题已经有人使用了很多假设进行研究分析。Major 将这个问题归结为一个 n 个资产的非网络资源分配问题（Major, 2002）。提出将攻击者和防御者脆弱性结合在一起，作为攻击者和防御者双方的资源 AA 和 DA 的函数，每个资产损坏造成的损失后果为 C：

$$V(AA_i, DA_i) = \{e^{(-AA_i DA_i)/\sqrt{C_i}}\} \left\{ \frac{AA_i^2}{AA_i^2 + C_i} \right\}$$

不幸的是，这个函数对于 $C_i = 0$ 和 $AA_i = C_i = 0$ 没有定义。此外，在万一攻击发生时，它不是基于成功的先验概率。但它是非网络世界中第一个将攻击者和防御者的脆弱性资源分配结合起来的效应模型。

Powers 和 Powell 扩展了 Major 的非网络资产风险模型，允许同时攻击多个资产，并制定如上的最高 - 最低目标函数（Powers, 2005; Powell, 2006）。原则上，这种方法导致一个螺旋式的军备竞赛，其中每个参与者依据其他参与者的响应增加其预算。我们已经说明，在一定条件下会产生军备竞赛收敛，导致了对于攻击者和防御者来说的优化分配。

这些模型对于非网络资产是适当的，如建筑物、桥梁、隧道以及其他结构，但并没有模拟出单个资产故障对整个系统造成的影响。网络系统必须考虑一次或多次节点或链路的攻击对网络

范围的影响。网络科学方法需要定义一个新的可以在网络资产上工作的风险和脆弱性模型。这个扩展导致了先前提出的 Al-Mannai-Lewis 模型。但是，Al-Mannai-Lewis 模型并不能反映全部状况，因为它的重点主要在于度序列分布。Al-Mannai-Lewis 攻击者 - 防御者风险问题给予节点太大的权重。除非链路也受到高度重视，否则 Al-Mannai-Lewis 军备竞赛算法很少将链路作为关键因素。正是基于这个原因，我们接下来分析关键链路。

11.5 关键链路分析

现在我们将注意力转到当节点遭受攻击时保护网络免遭破坏或不能运行的问题上来。为了便于讨论，假设当一个网络被分成各个组件使得一些节点或多个节点与其他节点间不可达时，该网络就变得*无法使用*。这可能代表电网中断，例如，当一个或更多的电力线路断开而不能传输电力。它也代表着航空公司航线、市郊铁路线或交通运输网络的航运路线的中断。链路的删除会因为失去连续性而中断网络，或者是因为将组件相互分离而暂停网络的运作。

*将链路弹性*定义为为了使网络分成各个组件所必须予以删除的链路的百分比：

$$\text{Link\_resilience} = \frac{t}{m}$$

其中 t = 损坏或删除的链路数，m = 初始的链路数。

显然，链路弹性值越高，就越难以分隔开一个网络，因此，其链路对攻击就更有抵抗力。一个链路弹性为 50% 意味着半数的链路都要被删除。在本节，我们探讨了随机、小世界和无标度网络对随机攻击链路的弹性，发现小世界网络是最有弹性的，随机网络其次，无标度网络弹性最小。

一个包含从源头流向接收节点商品的逻辑斯蒂网络，也容易受到链路攻击，特别是当去除一条链路切断商品流时更是如此。在这种情况下，即使网络没有被分离成单个组件，它也可能停止提供充足的商品流。在这个例子中我们将弹性定义为链路被阻塞或被删除时所能维持网络的最大流量的分数：

$$\text{Flow\_resilience} = \frac{c}{\max\_c}$$

其中，c = 源头到接收节点的流量，$\max\_c$ = 可能的源头到接收节点的最大流量。

再次，这种弹性测量范围为 0% ~ 100%——随着商品实际流的增加而增加。在本节中我们证明了当首先保护最大容量的关键链路，其次才是较低容量的路径时，保护链路发生的最优资源分配。我们提供启发式链路资源分配，使沿着最关键路径的这些流最大化。

另一种形式的弹性——*稳定性*对于网络安全也很重要。回顾一下，一个动态网络可能无法稳定，具体取决于它的下一个状态函数和拓扑。是否有可能通过删除某条链路使网络不稳定？对设计具有弹性的电网、交通系统和电子元件来说，这个问题很重要。没有稳定性弹性，我们就不能阻止因为恶意删除几条链路而造成的网络大规模不稳定。

*稳定性弹性*被定义为当删除链路时网络稳定的链路分数：

$$\text{Stability\_resilience} = 1 - \frac{s}{m}$$

其中 s = 如果删除造成网络不稳定的链路的数量，m = 链路总数。

我们研究前面已经定义和学习过基尔霍夫网络的稳定性弹性（参见第 9 章）。回想一下，基尔霍夫网络是有向网络，当节点流入的流量总和与流出的相等时该网络就是稳定的。问题是一条链路的去除如何影响到结果网络自身稳定并恢复运行的。鉴于删除的链路会产生一个不稳定的网络，从而也导出一个解——要么强化这些关键链路，要么添加冗余替代链路。

11.5.1 链路弹性

假设 G 是一个具有 n 个节点、m 条链路的任意拓扑结构的强连通网络。假设我们随机删除链路直到网络分成两个或两个以上部分为止。删除只有一条链路的节点的链路很容易将网络划分为两个部分。该网络目前分为两部分，一部分拥有（$n-1$）个节点，而另一部分只有一个节点。但如果没有这样的节点呢？此外，如果链路是随机被删除的？平均需要删除多少条链路才能将这样一个任意网络分成两部分？

*渗流*是一个添加链路直到形成一个巨大的连通组件为止的过程。*去渗流*恰好相反，删除链路，直到巨大的连通组件分隔成较小的组件为止。通常情况下，当删除单个节点的唯一一条链路时就会造成孤立，从而发生分隔。分隔成不同的组件所需删除的链路数取决于去渗流产生的链路弹性的估计值。

下面的 Java 方法实现了这个简单的过程。该算法非常简单——随机选择一条链路，如果该网络是连通的，就删除这条链路。如果该网络已经被分开，reply1. bit 将为真，通过两次计算节点将产生最大的组件——在链路的每个终端节点开始计数。方法 isComponent 遍历以 edges[e]. to 和 edges[e]. from 为根的生成树，同时在这两个方向上搜索完全生成树。如果搜索查找到所有节点，该方法返回 true。否则，返回 false。在这两种情况下，方法 NW_doDepercolate( )返回找到的最大组件的大小。该代码可在 NetworkAnalysis. jar 程序中找到，并使用 edges[ ] 代替 links[ ] 来存储网络链路：

```
class isComp {    //Return plex
    boolean bit = false;                          //Not a component
    int size = 0;                                 //Number nodes visited
}
public boolean NW_doDepercolate(){
     int e = (int)(Math.random()*nedges);       //pick any link at random
     isComp reply1 = isComponent(edges[e].to); //Count nodes twice
     isComp reply2;
     if(reply1.bit){
       NW_doCutEdge(e);
     }
       else {
          reply2 = isComponent(edges[e].from);   //Check both ends of link
          if(reply1.size < reply2.size) reply1 = reply2;
        }
        return reply1.bit; //Return size of largest
     }//NW_doDepercolate
```

方法 isComponent( )重置所有标志，启动向下递归过程来访问所有连接到初始节点的节点。使用了深度优先搜索，但也可以使用广度优先搜索。为什么？下一步，方法 isComponent( )计算了访问节点的数量并将它与网络的规模进行比较：

```
private isComp isComponent(int start_node){
       isComp reply = new isComp();
       reply.size = 0;              //Number nodes visited on each trip
       for(int j = 0; j < nnodes; j++){ //Reset flags
         nodes[j].visited = false;
       }
```

```
        visit_next(start_node);
        for(int j = 0; j < nnodes; j++){  //Reset flags
          if(nodes[ j] .visited) {
          reply.size++;
          nodes[j].visited = false;
          nodes[j].color = Color.red; //Visited nodes are red
        }
      }
      reply.bit = (reply.size == nnodes);
      return reply;
    }//isComponent
      //Recursive method
      private void visit_next(int j) {
        if(!nodes[ j] .visited){
          nodes[j].visited = true;
          for(int e = 0; e < nedges; e++){       //Push successors
          if(edges[ e] .to == j
                    &&
            !nodes[edges[e].from].visited)
               visit_next(edges[e].from);
          else if(edges[ e] .from == j
                    &&
          !nodes[edges[e].to].visited)
           visit_next(edges[e].to);
             }//for
          }//if
      }//visit_next
```

图 11-8 说明了方法 NW_doDepercolate( ) 在 n =200、m =1000 的中等大小的随机、小世界和无标度网络中的应用。使用程序 NetworkAnalysis. jar，去渗流数百个网络，并计数导致隔离需要删除的链路数量。这些数量被放置到不同 bins 中代表着删除的链路为 0 ~ 50，51 ~ 100，101 ~ 150，…，951 ~ 1000。删除的链路数与尝试总数的比例作为删除链路的函数提供一个分隔概率估计。经验数据是从模拟中得到的，标记为图 11-8 中所示的空心几何图形符号，而根据曲线拟合的估计值标记为实心符号。下一节中列出了一个近似估计方程，但很明显，链路弹性取决于网络拓扑结构。

仿真模拟的结果如何呢？类似密度的三个网络之间存在显著的差异。就链路弹性来讲，小世界网络最具有弹性，当 1000 条链路中的 60% 被删除时造成分隔。无标度网络是最没有弹性的，仅删除 45% 的链路就会使之分隔成不同的组件。随机网络居于中间，要求删除 55% 的链路才会造成分隔。从链路弹性方面来讲，这三类网络排名如下：

小世界网络的链路弹性最高

随机网络其次

无标度网络的链路弹性最低

这种非直觉的结果与直觉上的随机网络比小世界网络更具有弹性刚好相反，但事实上并非如此。为什么呢？

1. 无标度网络拥有最小适应性的原因显而易见，因为它包含了许多低度的节点。因此，需要删除较少的链路就可以断开网络。选择连接到 hub 节点的一条链路的概率会较高，但如果在该链路另一端的节点的链路较少时概率也会较高。低度节点和 hub 数目之间的这种不平衡导致了较

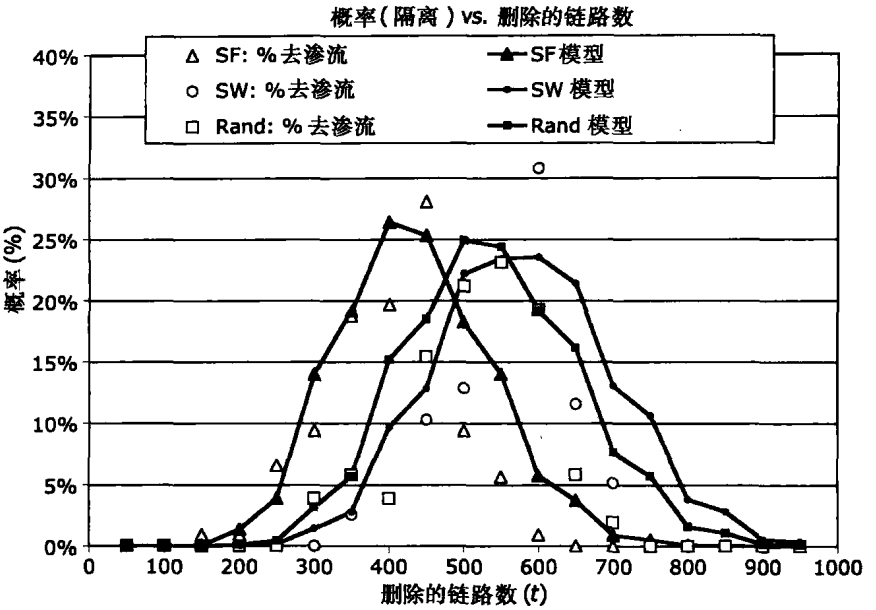


图 11-8 随机、小世界和大小适中的无标度网络的去渗流优化结果，$n=200$，$m=1000$，重联概率 $p=5\%$，$\Delta m=5$，并且 $\lambda=10$。实线是基于课文中导数的模型。对于小世界网络参数 $\alpha=0.42$，对于随机网络为 0.50，对于无标度网络为 0.75

低的链路弹性。很简单，无标度网络具有太多低度的节点。

2. 比起随机网络，在一个小世界网络中删除一个低度节点的链路的概率更低，因为它的度序列分布范围很窄，这意味着大多数节点具有平均数目的链路。很少节点具有较低的度。度的分布（从而导致链路）在随机网络中就会更广泛，这意味着随机选择的链路更有可能是连接到一个较低度的节点上。换句话说，我们加倍了将节点从网络中分离的机会，因为每个链路都有两个终端节点。这种加倍的危险致使网络隔离的机会增加。对于有广泛度序列分布的网络来讲，风险更高，这会导致随机网络更早地隔离。因此，小世界网络比随机网络更富有弹性。但是这种差异会减小，因为一个小世界网络的熵会由于重联概率的增加而增加。

3. 图 11-8 表明，隔离概率是删除链路的数量的函数（因此导致链路弹性），服从二项式分布或泊松分布。如果属实，这对于随机类网络来讲并不感到意外，倒是对于小世界网络来讲很有些意外，但对无标度网络则完全违反常理。一般来说，随机删除链路就像反向的 Erdos-Renyi 生成过程。随机删除链路的结果就会造成随机化，因为它往往会通过引入熵来重组任何网络。随机删除的链路越多，熵就越多，就会导致某种随机的度序列分布。换句话来讲，随机删除链路会造成将度序列分布重组成类二项式分布。但是，我们将会看到，它不是一个纯粹的二项式分布。

我们在下一节推导出一个链路弹性的近似值，并表明去渗流重组是相似的，但并不等同于纯粹的随机分布度序列。两者是类似的，因为去渗流将熵注入到网络中。结果证明它们是不同的，链路的删除会同时改变两个节点的度——在链路的每端各一次。一次同时更改两个节点并不是一个纯粹的泊松过程。实际上，如图 11-8 所示的实线并不服从泊松分布。

11.5.2 链路弹性模型

假设泊松分布是设计逼近图 11-8 的隔离分布概率的合理的出发点，因为去渗流类似于产生随机网络的 Gilbert 生成过程。我们通过两种方式使用纯粹的泊松过程建立隔离模型：

1. 设 α 是一个常数，用于描述去渗流网络中度序列的均匀性偏差：$0\leqslant\alpha\leqslant1$。

2. 设隔离概率为删除每条链路的两端的 stub 所产生的结果，通过改变删除链路的两端的度

序列分布来改变度序列分布。这个概率会受到不论删除链路的哪端都可能造成把网络隔离成组件的影响。

删除 t 条链路后，使平均度和选择属于删除链路的节点的概率分别如下：

$$\lambda = \frac{2(m-t)}{n}$$

$$p = \frac{2\alpha\lambda}{(m-t)} = \frac{4\alpha}{n}$$

其中，m = 原始链路数；n = 节点的数目；t = 删除的链路数；α = 将要确定的依赖于度序列的常数；p = 选择平均度为 λ 的节点的概率。

选择具有平均度为 λ 的节点的概率是通过关注删除 t 条链路后留下（$m-t$）条链路获得的，并且一端节点属于对应选择链路的概率为 $\lambda/(m-t)$。因此，两个端节点属于选择链路的概率为 $2\lambda/(m-t)$。参数 α 将在后面通过曲线拟合来确定。

在 t 次尝试中选择随机链路是一个参数为 t 、λ、p 的泊松分布过程。因此我们使用二项式分布来模拟在 t 次尝试中选择 λ 条链路连接到度为 λ 的节点的概率。所有 λ 条链路都被删除后，该节点就成为孤立的。随机链路选择删除了 t 条链路，但影响到 $2t$ 个节点。一个特别的节点在删除 λ 条链路后就会孤立，但链路删除对链路两端都有影响。因此，二项式分布代表了删除链路时删掉两个特殊节点间的链路的概率。这就是为什么我们设置 $p = 2\lambda/(m-t)$ 而非 $\lambda/(m-t)$ 的原因。

使用组合函数和向上舍入，我们得到：

$$B(t,\lambda;p) = C_\lambda^t p^\lambda (1-p)^{t-\lambda}$$

$$C_\lambda^t = \frac{t!}{(t-\lambda)\lambda!}$$

其中

$$\lambda = \text{roundup}\left(\frac{2(m-t)}{n}\right)$$

尽管 B 为选择两端节点的概率，但它并不是分隔网络的概率。我们考虑了所有可能的事件以获得隔离概率，在链路的一端或在另外一端分隔或在两端同时分隔。利用我们在图 11-1b 中使用过的德摩根定律，但是使用两个而不是三个错误，包括曲线拟合参数 α，我们就得到

$$\text{Prob(separation)} = 1 - [(1-B)(1-\alpha B)] = B(1+\alpha-\alpha B)$$

其中，α = 网络类别参数；B = 修正过的二项式分布函数。

这种结果是通过考虑了两个错误获得的：一个是以概率 B 发生的，而另一个发生的概率是 αB。例如，如果 $\alpha = 1$，在链路一端或两端隔离的概率为 $B(2-B)$。如果 $\alpha = 0$，概率就为 B。参数 α 代表了一个典型的节点不是很典型——其度与平均度 λ 不同。实际上，在一个小世界网络中参数 α 应该是很低的，因为节点度的变化是很小的，相应地，在无标度网络中由于节点度的偏差是很高的而使 α 很高。实验支持了这种说法。

现在我们有一个 t 条链路删除后的完整隔离概率模型：

$$\text{Prob(separation:}n,m,t,\alpha) = B(1+\alpha-\alpha B)$$

$$\lambda = \frac{2(m-t)}{n}$$

$$p = \frac{4\alpha}{n}$$

$$B(t,\lambda;p) = C_\lambda^t p^\lambda (1-p)^{t-\lambda}$$

$$C_\lambda^t = \frac{t!}{(t-\lambda)\lambda!}$$

其中

$$\lambda = \text{roundup}\left(\frac{2(m-t)}{n}\right)$$

$$\alpha = \text{网络类别参数}$$

参数 α 对于图 11-8 中的经验数据更改如下：

小世界网络 $\alpha = 0.42$；

随机网络 $\alpha = 0.50$；

无标度网络 $\alpha = 0.75$。

这些结果支持用 α 代表这里研究的各类网络的度序列分布的变化。α 对图 11-8 中曲线形状的影响，就是向左边或右边移动分布，对应于典型节点的平均度变化。但平均度 λ 对 $n = 200$、$m = 1000$的所有网络都是相同的，那么为什么会有这种移动？随着链路被删除，平均度会减少。随着它的减少，隔离概率会移向左边——朝向原来的图。无标度网络要比其他网络变化更大，所以它的隔离分布要比其他两类更偏向左。因此，无标度网络的 α 是三类网络中最大的，代表着平均度中最大的移动。

随着链路弹性的增加，参数 α 减少，所以它也与弹性相关。我们可能会使用（$1-\alpha$）作为衡量弹性的方法，但它确实不是线性形式。此外，弹性加倍并不会造成（$1-\alpha$）的减半。此外参数 α 并不明显地与删除的链路相关。基于这些原因，我们不考虑将它作为衡量弹性的测量。

这里导出的近似很粗略。我们将寻找更好的近似的任务留给读者去完成。总之，链路的删除倾向于使网络随机化，增加它的熵，降低其密度。该网络在某一密度阈值最终分成组件，这具体由拓扑结构来决定。我们发现与随机和无标度网络相比，小世界网络仍然以很低的密度保持不变。这也许是由于底层的 k - 规则网络所固有的高度聚类所造成的。另一方面，无标度网络在高密度下隔离，可能是因为它们从具有太多低度的节点开始。

11.5.3 流弹性

至此，我们已经假设将网络表示成一个静态系统。在许多现实世界的应用中，网络代表了通过一个系统的商品流。例如，一个供水系统的网络模型具有带方向性的流，电信网络具有双向的流，电力网可能会在一段时间内支持一个方向的流，然后在另外一段时间流向相反。这自然引起了有向流系统的脆弱性问题。

如果网络表明了从源到接收节点的流，那么很明显源节点非常重要，因为没有它们，就没有流！显然，我们应该保护源节点，而忽视它们的度或费用。此外，如果网络中某条链路断开，那么全部的链路子图和节点就失去了商品。直观地讲，我们希望“上游”节点和链路比“下游”节点和链路更重要，而它们是否具有优先权要取决于保护它们的成本。在下面我们提出了一种启发式的涌现过程，以便找到将有限的资源最优分配给节点和链路，使得流向接收节点的总商品量最大化。

流优化是运筹学中的一个老问题，所以该问题应该很容易解决。但是，下面的问题并没有已知的闭合形式的解，因为它提出了：“链路和节点如何进行最优资源配置，才能使流动到接收节点的期望值最大化?”这一问题带来了一些分析性的挑战，正如我们将要说明的那样。再次，图 11-1a 的简单网络被用来说明这种技术。

考虑图 11-1a 中例子的有向流网络的额外属性如下。假设节点/链路容量的后果等同，用 C_i 表示；$\max DA_i$ 等于可删除脆弱性的代价；v_i 等于脆弱性；g 像前面一样等于度。为了简化标记符号，将可用性 a_i 定义为每个节点/链路不会失败的概率：

C_i 由于节点/链路 i 故障带来的损失或后果；

$\max DA_i$ 完全删除脆弱性需要的资源；

a_i $(1-v_i)$：节点/链路不会故障（可用性）的概率；

f_i 通过一个节点的商品流 $0 \leqslant f_i \leqslant C_i$；

h_i 通过一条链路的商品流 $0 \leqslant h_i \leqslant C_i$；

$w_j \xrightarrow{\text{flow}} h_i$ 从节点 w_j 到链路 h_i 的商品流；

$w_j \xrightarrow{\text{flow}} h_i$ 从链路 h_i 到节点 w_j 的商品流；

n —— 节点数；

m —— 链路数。

接下来，将在每个节点/链路上的流量定义为其容量或来自进入链路/节点输入总和的加权最小值：

$$\text{链路：}\quad h_i = a_i[\min\{C_i, f_j\}]; w_j \xrightarrow{\text{flow}} h_i$$

$$\text{节点：}\quad f_i = a_i\left[\min\left\{C_i, \sum_{w_j \longrightarrow h_i} h_j\right\}\right]$$

换言之，通过一个节点/链路的商品量受限于它的容量，以及有用性 a_i 乘以流入量的乘积。期望流量是可用性乘以实际流量。在某种意义上来讲，期望流量是通过一个节点/链路的某种等级的商品流的可能性。对于链路，这意味着期望的商品量等于其期望的源节点流或预期的最大后果。对于节点，这意味着期望的流量是期望的进入链路的流总和，或期望的最大后果。

接收节点流量就是进入总和或最大容量乘以可用性。源节点没有到达链路，所以它们的流量由节点的可用性和最大容量决定：

$$\text{接收节点：} f_i = a_i\left[\min\left\{C_i, \sum_{w_j \longrightarrow h_i} h_j\right\}\right]$$

$$\text{源节点：} f_i = a_i C_i$$

一个网络的总输出等于流向所有接收节点的总的流量 F。期望的总输出等于所有流向接收节点的流量的总和。这个总和并不受接收节点容量或可用性的限制，因为我们不希望接收节点对最大流量有总体的限制影响。此外，请注意从每个内部节点输出的流量总和可能超过内部节点的容量——从而违反了**基尔霍夫定律**。当一个节点有不止一条外出链路，而每条链路承载量等于节点容量时，这种超出就可能会发生。例如，节点 w 带有后果为 100，分别在两条外出链路中的每一条传输 100 个单位，因此它总共传播了 200 个单位。这意味着，总输出可能超过所有源节点的总供给流量。我们之后再回到这个问题，但是为了最大流量分析，让 F 等于到达所有接收节点的所有流量总和。F 是我们想要最大限度地达到预算 B 的目标函数：

$$F = \sum_{s=1}^{k} w_j = \sum_{s=1}^{k} a_s\left[\min\{C_s, \sum_{w_s \longrightarrow h_i} h_j\}\right] \text{受限于} \ 0 \leqslant a_i \leqslant 1$$

假设可用性和资源费用之间呈线性关系，我们将可用性等同于资源分配如下：

$$a_i = \frac{DA_i}{\max DA_i}$$

最后，资源是有限的，因此我们另外增加一条约束：

$$\sum_{i=1}^{n+m} a_i = \sum_{i=1}^{n+m} \frac{DA_i}{\max DA_i} \leqslant B$$

给定预算 B，目的是为了找到 a_i 使得流量总和 F 最大化：

$$\max_{a_i}\left[F = \sum_{s=1}^{k} a_s\left[\min\{C_s, \sum_{w_s \longrightarrow h_i} h_j\}\right]\right]$$

$$\sum_{i=1}^{n+m} a_i \leqslant B$$

这种优化问题乍一看似乎很简单。像我们早些时候为线性分配算法所做的，使用拉格朗日乘数方法，导出系统的非线性方程形式 $xyz - \lambda = 0$。如果没有比上述规定的流量模型更多的限制的话，这种非线性系统没有唯一解（x,y,z）。数学的联立方程组是不确定的，所以唯一解是不能实现的。事实上，也不能保证存在一个唯一解——读者可能会找到图 11-1a 中的一个以上的网络最大流量，但分配给每个节点和链路以不同的资源。这导出通过涌现过程计算最优解的启发式方法。

11.5.4 流启发式的 Java 方法

网络流量的计算是从确定接收节点开始的，然后通过网络递归地回溯到源节点，直到所有的节点和链路都遍历过为止。这是由一个根方法和两个递归子方法组成的（这两个子方法一个用于节点，另一个用于链路）。方法 NW_doFlow( )假定节点此前已标记为红色（接收）、绿色（源）、黄色（已访问）或白色，分别表明它们的状态为接收、源、已访问或内部节点。如果它找到红色（接收）节点，并沿着接收路径从接收节点回溯到源节点，计算每个沿回溯路径的节点/链路流量。避免重复循环，只要访问过的节点就标记为黄色。NW_doFlow( )返回所有接收节点的总流量。

```
public double NW_doFlow(){
  double totalFlow = 0;
  NW_RestoreNodeDegree();          //Restore color (White) and node degree
  for(int i = 0; i < nnodes; i++){      //Backtrack from sink (red) nodes
    if(nodes[ i] .color == Color.red) {
       totalFlow += nodeFlow(i);           //Objective function
  }
}
   return totalFlow;
}//NW_doFlow
```

方法 nodeFlow 和 linkFlow 为每个节点/链路返回以后果 C_i 为单位的商品流量。请注意，每个节点/链路都有一个相关的流量、失败的概率和后果 c。这些程序变量与数学公式的匹配方式如下：

nodes/edges. flow： f_i

nodes/edges. ProbOfFailure： v_i，这里 v_i 是脆弱性

nodes/edges. maxDA： $\max DA_i$

nodes/edges. C： C_i

方法 nodeFlow 调用方法 linkFlow，然后 linkFlow 调用 nodeFlow 等。因此，这两种方法递归地从接收节点回溯到源点，在节点和链路之间交替，直到找到源点为止。从图论来讲，NW_doFlow( )构造一棵生成树，从源点达到接收节点。例如，图 11-1a 网络的生成树包含了所有的节点和链路，除了节点 1 和节点 3 之间的链路 2。该网络是连通的，所以每一个节点从每一个接收节点都是可达的，但没有必要将所有链路列入生成树中。

```
//Recursively calculate flow
private double nodeFlow(int node){
   double flow = 0;
   if(nodes[ node] .color == Color.yellow)       //Already visited node
     return nodes[ node] .flow;
   if(nodes[ node] .color == Color.green){       //Source node
```

```
        flow = nodes[node].C*(1.0-nodes[node].ProbOfFailure);
        nodes[node].flow = flow;
        return flow;
       }
        if(nodes[ node].color == Color.white)
        nodes[node].color = Color.yellow;       //Mark it as visited
        for(int j = 0; j < nedges; j++){        //Process all incoming links
        if(edges[ j].to == node){
          flow += Math.min(nodes[node].C, linkFlow(j))*(1.0-
nodes[node].ProbOfFailure);
            }
      }
      nodes[node].flow = flow;
      return flow;
     }//nodeFlow

     //Recursively calculate flow
      public double linkFlow(int link){
       Edge e = edges[link];
       double flow = Math.min(e.C, nodeFlow(e.from))*(1.0-e.ProbOfFailure);
       e.flow = flow;
       return flow;
      }
```

11.5.5 网络流资源分配

由于在流网络中优化资源分配缺乏一个闭合形式的解，这迫使我们只能使用经验方法。我们研究了启发式的性质，可以通过涌现尽快找到最佳的解决方案，但并不能保证每次都是最优的。相反，流优化的涌现启发式方法在大多数时候都能找到一个最佳的解。如果我们多次运行涌现过程，并在结果中选择最佳的解，就可以得到我们认为的最好的分配。但是，我们不能保证这是最好的！

像在本书中所有的涌现过程一样，最佳流量涌现过程非常简单。最初，预算分布在最大流量路径上，由有效度量确定：$\min C_k/\max DA_i$，其中 $\min C_k$ 是所有接收节点到源节点路径上找到的最小值后果。我们通过设置可用性为 100% 获得 $\min C_k$，计算最大流量，然后排序接收节点。接收节点 k 的流量刚好是 $\min C_k$。

从最高流量接收节点开始，沿着由最大化 $\min C_k/\max DA_i$ 定义的路径回溯，直至到达源节点为止。然后，逆向进行这个过程，从源节点到接收节点，沿最大 $\min C_k/\max DA_i$ 路径尽可能多地分配预算到节点和链路上。这个初始阶段的逻辑是最大化 $\min C_k/\max DA_i$ 路径能更好地利用资源。但是，正如这个简单的例子所表明的，情况并非总是如此。因此，启发式必须利用第二个阶段监视防止这种违反直觉的方案。

初始阶段之后紧跟着一个广泛的涌现阶段，即在节点/链路对之间交换一个小的可用性量，输出流量重新计算并与前面的输出相比……直到所有的节点流量达到最大值为止。这个过程无限地持续下去，或直到看起来流量的进一步增加无法获得为止。这一阶段的逻辑是最大的流量将从小的改变并经过大量的时间而涌现，因为成功的交换会被保留下来，而不成功的交换则会被拒绝。

最佳流量涌现

1. 最初，对所有的节点/链路设置 $a_i = 1$，并计算最大可能的流 F_{max}。这将在接收节点间建立

一个排名，从最高流量到最低流量：$f_{j_1} \geqslant f_{j_2} \geqslant \cdots f_{j_k}$。这也决定了 $\min C_k$：$\min C_{j_1} = f_{j_1}$，$\min C_{j_2} = f_{j_2}$，…，$\min C_{j_k} = f_{j_k}$。

2. 将接收节点涂为红色，源节点涂成绿色。重置所有节点/链路的可用性为零：$a_i = 0$。可用的预算设置成 $b' = B$。

3. 第一阶段：重复直到所有节点 $s = 1, 2, \cdots, k$ 已处理，或现有的预算已经达到零（$b' \leqslant 0$）为止：

   a. 从接收节点 s 到源节点 z 查找最大的路径，按照最高的效率比选择每个节点/链路：$\min C_s / \max DA_i$；也就是说，按照最高 $\min C_s / \max DA_i$ 路径从接收节点到源节点。

   b. 从 3a 步骤中定义的源节点 z 开始，沿最大路径返回接收节点 s，沿路径分配 $DA_i = \min(b', \max DA_i)$ 到每个节点/链路。

   c. 从可用预算中减去每个分配：$b' = b' - DA_i$。

4. 第二阶段：如果还有剩余预算 b'，将它均匀地分布在所有节点/链路上，不大于 $\max DA_i$。

5. 无限重复：

   a. 随机选择一个 DONOR 节点/链路和一个 RECIPIENT 节点/链路。交换一个单位资源：从 DONOR 取出一个单位 DA_{DONOR} 转给 RECIPIENT。

   b. 计算总流量。

   c. 如果总流量减小，反向进行 DONOR/RECIPIENT 的分配：将一个单位返回给 DONOR 并恢复 RECIPIENT。否则，什么也不做。

表 11-4 经过流分析的资源分配

节点/链路	C	$\max DA$	DA	脆弱性（%）	流	风险
节点 1	10	15	8.5	43	5.7	4.3
节点 2	20	5	0.0	100	0.0	20.0
节点 3	10	10	8.5	15	4.8	1.5
节点 4	5	5	0.0	100	0.0	5.0
节点 5	15	1	1.0	0	4.8	0.0
链路 1	10	1	0.0	100	0.0	10.0
链路 2	10	1	0.0	100	0.0	10.0
链路 3	10	1	1.0	0	5.7	0
链路 4	10	1	0.0	100	0.0	10.0
链路 5	10	1	1.0	0	4.8	0.0
总和	110	41	20.0	—	4.81	60.8

该算法在程序 NetworkAnalysis.jar 中采用名“Max Flow”来实现，为图 11-1a 中预算为 20 单位的简单网络产生分配，如表 11-4 所示。例如第 1 阶段，确定了第一个最大路径，从节点 5 通过链路 5 至节点 3，然后链路 2 到节点 2，最后是链路 1 到节点 1。事实证明，这不是产生最佳流量的路径，而只是初步的分配。在第 1 阶段，接收节点 4 和 5 的 $\min C$ 分别为 10 和 5。节点 5 被选定作为出发点，因为对于节点 5，$\min C_5 / \max DA_5$ 是最大的：10/1 = 10，节点 4 为 5/5 = 1。

沿着链路 5 至节点 3，然后链路 2 到节点 2 产生了最大路径，因为 $\min C / \max DA$ 对于链路 2 为 10，节点 2 的 $\min C / \max DA$ 为 4。继续沿着链路 1，第一阶段到达源节点 1 后就停止。沿着这条路径回溯，为节点 1 分配 15 个预算单位，为链路 1 分配一个单位，为节点 2 分配 5 个单位，为链路 2 分配一个单位，以此类推，直到预算递减到零，或到达源节点 5 为止。如果有足够的预算，沿着从接收节点 4 到源节点 1 的最大路径执行同样的分配。

结果证明，对于20个单位预算，最优分配排除了链路2和1以及节点2（见表11-4）。第二阶段的算法接管，并逐步从两条链路和节点转移资源，因为分配给最短（直接）路径从节点1，通过链路3到节点3，链路5到节点5，而使流量增加。因此，节点1、3、5和链路3、5接收非零分配，从而建立一条从源节点到接收节点的最大流量路径。

方法NW_doFlow()使用表11-4中的分配向量*DA*返回了4.81个单位的流量。这是20个单位预算的最好可能的流量。由于链路非常符合成本效益，所以给它们分配全部的数额，将它们的脆弱性减少为零。下一步，在预算允许的情况下尽可能保护源节点和接收节点，同时确保中间节点3的安全。

首先，请注意，此资源分配向量是非常不同于通过线性和攻击者－防御者算法得到的分配向量的。最大流量将风险降低至60.8个单位，而线性分配策略将风险降低到16个单位。流量优化沿着一条路径分配资源，而不是集中到关键节点。其实，流量优化沿着关键路径最大限度地分配了有限的资源，以最低的后果与最高的脆弱性消除成本max*DA*之比来定义。涌现启发式选择了一个包含最高效益率的路径而不是最低风险的路径。

流量优化根据最大流量而不最小风险定义了脆弱性。这与减少风险的目的相背离。也就是说，流量最大化可能导致高风险的网络，因为这两个的目标函数是不同的。流量最大化稀释了资源效果，因为它沿着一条路径铺开来，导致了更高的网络风险[⊖]。

最后，涌现启发式没有界定DONOR和RECIPENT之间交换的资源单位的尺度。这很成问题，因为一个小的设置如0.01会容易导致一个虚假的最大值。如果时间步长变化太小，涌现过程在到达最大化过程中就无法避免这个漏洞。另一方面，太大的交换单位由于低的分辨率很容易在没有到达最大流量时就停止——坑穴太小而难以陷入！运行NetworkAnalysis.jar，初始步长值为0.1，然后在5000次迭代后将它降低到0.01产生最大流量4.81，如表11-4中所示。需要超过20 000次迭代才能找到最大流量！

交换单位（在程序NetworkAanlysis.jar中称为Dollar_Step）应先定得很高。最好近似设置为最大max*DA*值的约1%。随着涌现接近最大奇点，应该减少交换单位。这才有可能使过程得以求解目标函数（流量）的微小变化，并求出最大流量的峰值。NetworkAnalysis.jar程序中允许的最小交换单位为0.01。

总之，这里描述的涌现过程基于两个主要条件，当然这还有待于证明：

1. 资源分配沿着最有效的资源利用率路径，min*C*/max*DA*。第一阶段通常“接近”最佳的解，从而避免可能阻止进一步增加到真正的最大流量的假的最大值。

2. 随机交换少量的资源最终导致一个最大的固定点。这种说法没有得到证实，但在实践中，如果交换单位选择得合适，大多数时候会随机达到最大固定点。但这只是一点有关如何选择交换单位的指导。这仍旧是一个开放研究的问题。

尽管存在这些弱点，涌现过程找到了最大流量分配，利用最有效的方式分配有限的资源。实际上，这可能需要很多实验，在几百个节点的适度网络中找到最大固定点。每次实验可能会占用大量计算时间，但如果答案很重要，花费的计算时间也是值得的。

11.5.6 结构化网络中的最大流量

最大流量是否取决于网络拓扑？假设我们控制所有的变量恒定，如网络大小为n，链路数为m，节点/链路属性为C和max*DA*，以及各种不同的网络拓扑。使网络的拓扑为随机的、无标度的或小世界的。此外，让预算变化，以便我们可以观察每类网络如何利用资源。

⊖ 这是一个只要资源沿着路径分散而我们定义网络风险的人造制品。

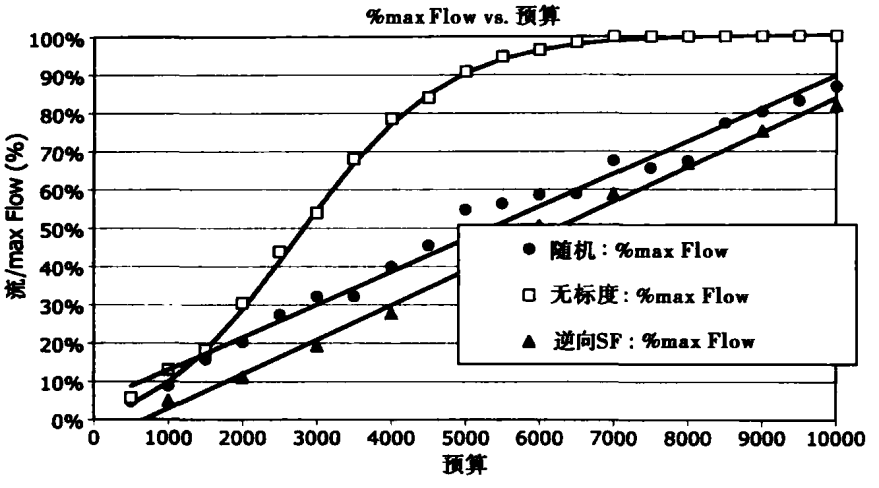


图 11-9 $n=50$，$m=100$（$\Delta m=2$），$C=100$，max $DA=100$，对于所有节点和链路，最大可能流量与资源预算的百分比

图 11-9 中展示了以最大流量通过一个随机和两个无标度网络时对应预算的优化分配的结果。随着预算的增加，我们期望流量等级也有所增加。但是，增加的形状很惊人。随机网络的关系是线性的，包含一个源（hub）和许多接收节点（刚好与无标度相反）的无标度网络也是线性的。令人惊讶的是，对于包含一个接收节点和多个源节点的无标度网络，这种关系是非线性的。

ER（Erdos-Renyi）生成过程产生了一个带有均匀分布的源和接收节点的随机网络。仍旧预期路径长度和以 $C/\max DA$ 表示的成本效益大致是相同的。因此，流量到达接收节点的比例沿着一条直线进行。提供的预算越多，就有更多的商品流。这符合直观的看法。

这两个无标度网络的流量与预算的曲线是非直观的，并且它的形状是不可预料的。用于产生无标度网络的 BA（Barabasi-Albert）生成进程，将所有添加的链路指向网络。这将产生一个有许多源而只有一个接收节点（即网络的 hub 节点）的有向网络。如果我们逆转所有链路的方向，那么该产生一个有许多接收节点而只有一个源节点的无标度网络。

图 11-9 显示了两种结果：无标度和逆向无标度。无标度网络有许多源但只有一个接收节点分配资源，使流量遵循一条逻辑斯蒂曲线，而逆向无标度网络（一个源点，多个接收节点）会产生类似随机网络的线性关系。换言之，当有很多输出节点时，资源优化分配以最大化流量生成一个输出线性增加，但如果只有一个输出节点，就是一条逻辑斯蒂曲线。

为什么单个接收节点的无标度网络服从逻辑斯蒂曲线？这是一要求很高而资源供给有限的经典的系统例子。换句话来讲，当只有一个输出节点存在时，只有很少的节点会分配到资源。起初，分配给流量的商品百分比率剧增，但由于越来越少的节点能够承受流量的增加，回报沿着平坦的 S 曲线降低。如下公式对教科书中内容进行非常好的曲线拟合：

$$\frac{F}{\max F}=\frac{1.07}{(1+15e^{-B/1000})}-\frac{1.07}{1+15};B=1000,2000,\cdots$$

例如，对于图 11-9 中定义的无标度网络参数以及预算 $B=5000$，获得的最大流量的百分比为 90.5%：

$$\frac{F}{\max F}=\frac{1.07}{(1+15e^{-5000/1000})}-\frac{1.07}{1+15}=0.972-0.067=0.905$$

随机网络线性地利用资源，产生最大流量；其他结构化网络（如这里研究的无标度网络）非线性地利用资源。缺少接收节点会产生一个切断流量进一步增加的瓶颈，而不管分配了多少资源。逻辑斯蒂曲线与资源预算很好地服从回报递减定律。

小世界网络没有源点或接收节点，但如果我们在小世界网络中插入一个源点和接收节点，它也将表现为非线性。小世界的流量主要取决于一个源节点/接收节点如何靠近于捷径链路。此外，节点的增加会导致一个供应短缺/瓶颈，类似只有一个接收节点的无标度网络，这意味着预算和最高流量之间的关系将是非线性的。小世界网络中的最大流量分配问题的探索留给读者作为练习。

11.6 基尔霍夫网络的稳定性弹性

在许多网络科学对现实世界系统的应用中，保障网络以及节点和链路的稳定性很重要。例如，电路、电网及带有反馈链路的机械控制系统，不会因为一条或多条链路的删除而断开，但它们可能会因为某条链路的删除而不稳定。潜在的不稳定因素几乎隐藏在任何包含某些商品流（如水、电子、网络分组以及生物系统中的信号）的网络表面之下。因此，这种形式的弹性是不容忽视的。

我们在第9章介绍了闭合系统的基尔霍夫网络。这些网络在一个没有源节点、接收节点的系统中模拟了有向流量。很显然，在一个有向网络中，如果源或接收节点从网络中断开，流就很容易中断。但由于去除某些内部链路而造成潜在的不稳定因素，尚未在文献中得到充分的研究。我们表明，删除单条链路可能因为基尔霍夫流量的不稳定而导致整个网络的破坏。一旦不稳定，基尔霍夫网络就不能恢复，除非网络中其余节点/链路同步的充分必要条件得到满足。

回顾第9章中内容，如果一个基尔霍夫网络同步会有以下两个条件：（1）它的基尔霍夫特征值小于1；（2）它包含了长度相对互质的循环。一个不稳定的基尔霍夫网络的简单例子就是任何规模大于2的环形网络。如果基尔霍夫网络有一个固定源节点，它就不能恢复稳定，但如果链路删除导致产生一个接收节点（不是源节点）它就可能恢复稳定。我们在这个分析中不考虑一条链路的删除导致一个源节点的产生的情况，这种特殊情况留给读者作为练习。

重新考虑第9章中图9-8所示的Pointville电力网络。因为由Calvert PWR→SS5→Minor和Calvert PWR > SS4→Minor所提供的冗余线路，删除Calvert PWR到Minor的链路（链路5）并没有阻塞系统的流量。事实上，如果我们考虑反馈链路13（从负载到发电机）作为一条控制链路而不是电力线路，然后对于所有节点和链路用1000个单位预算提供99%的最高电力流量即后果 = 100、maxDA = 50进行流量优化。链路5的删除对流具有很小的影响——在相同条件下1000个预算单位仍旧保护着99%的最大电力，只是电子流通过冗余路径。但是，链路5的删除造成了网络的不稳定，它无法同步！这意味着，基尔霍夫网络的不稳定造成电网的故障。

删除以下任何一条链路都会造成Pointville电网的不稳定：链路1（在发电机和Calvert PWR之间），链路5（Calvert PWR到Minor）或链路10（Minor到Load）。这些链路的删除对网络流量没有影响，但它们对网络的稳定性至关重要。换言之，删除单条链路造成的容量损失并没有失去稳定性那么重要。链路弹性高，但稳定性弹性较低。

Pointville电网还解释了关键链路的另一个定义。这种链路的确定是通过计算网络的基尔霍夫特征值，以及在删除每条链路后网络剩余的环长度来标示的。如果链路删除造成基尔霍夫特征值大于1，或任何两个环没有相对互质的长度，那么该链路对于稳定性是关键的。这是确定任何网络链路稳定性的方法。

下面的关键链路分析算法“删除”每条链路，每次一条，并计算基尔霍夫特征值和网络中剩余循环的环长。如果特征值大于1，或任何两个环没有相对互质的长度，该网络变得不稳定。如果一个或两个上述条件存在，该链路标记为“关键的”。测试每条链路，如果发现有一条是关键的，那么整个网络很容易受到删除链路的影响。

程序NetworkAnalysis.jar将关键链路标记为红色，表明删除该链路网络会不稳定。以

Pointville 电网为例，NetworkAnalysis. jar 发现三条重要链路：链路 1（发电机到 Calvert PWR），链路 5（Calvert PWR 到 minor），或链路 10（minor 到 load），并标记为红色。因此，3/14 = 21.4% 的链路是关键的。Pointville 电网的稳定弹性为 11/14 = 78.6%。

关键链路分析

对网络中的每条链路 L 重复进行以下步骤：

1. 将链路标记为“删除”。使 $q=1$，如果没有环或环具有相对互质的长度。
2. 计算基尔霍夫特征值 ε_k，以及没有标记的链路 L 的网络的环长度。
3. 如果没有环，设置 $q=1$；否则设置 $q=0$。如果有环并且任何两个环都有相对互质长度，设置 $q=1$。
4. 如果 $\varepsilon_k > 1.0$ 或 $q \neq 1$，L 涂成红色并标记为关键。

这种算法会耗费时间，因为它计算一个特征值和长度可达 n 的循环长度——通过将系统矩阵 B 提高到每条链路有用性为 B^n。矩阵乘法需要 $O(n^2)$ 的计算，B^n 和特征值的计算需要 $O(n^3)$ 的计算，因此，一个具有 n 个节点和 m 条链路的网络需要 $O(mn^3)$ 的浮点运算！

奇怪的是，随机网络包含许多环，因此任何两个环的长短相对互质的可能性很大。因此，随机网络本质上是稳定的！例如，一个 $n=50$、$m=150$ 的随机网络可能含有 50 ~ 100 个循环。一个循环只含有长度 2 或 3 以达到相对互质的条件是很必要的，因为 2 和 3 是互质数。

无标度网络通常有少数环，但它们包含许多源节点和接收节点。因此，无标度网络本质上是不稳定的。重联网络以便消除源和接收节点而引进了环，很有可能有两个或更多的环其长度互质，这意味着在一个闭环的系统中无标度网络有可能同步。因此，一个循环无标度网络具有少量关键链路。

最后，小世界网络没有源节点和接收节点，但它们有很多环，因此它们很可能会同步，从而稳定。仅添加几条随机链路有可能使得环长度为互质数。因此，小世界网络就像随机网络一样倾向于稳定——它们没有关键链路。

关键链路存在于如电力网络的结构化流网络中。这不仅是一个理论，2003 年电网停电影响了美国和加拿大的 5000 万居民，或许这只是因为俄亥俄州的某个链路的断开而导致的。1997 年，俄勒冈州和加利福尼亚州之间交界的某个连接线路故障断开，多个小时后美国西部地区的 11 个州和加拿大西部的部分州停电。即使电力公用事业公司运行了复杂的建模和仿真软件来预测和控制整个电网的电力高峰，这些故障还是发生了。难道网络科学掌握着理解这种复杂系统的关键？这留作读者练习。

练习

11.1 假设图 11-1a 中的简单网络包含的节点和链路具有相同的后果（等于 2）及相同的脆弱性缩减费用（等于 1）。对于以下情况，预算为 5 时的优化防御者分配模式是什么和标准化的网络风险是多少？

(a) 使用随机分配策略；

(b) 使用线性脆弱性缩减分配；

(c) 使用指数脆弱性缩减分配；

(d) 使用最大流优化启发式。

11.2 使用练习 11.1 中的同一网络，并且攻击者预算为 5 个单位，防御者预算为 6 个单位，对于以下条件，资源的优化分配和标准化的网络风险是多少？

(a) 随机防御者，随机攻击者；

(b) 线性防御者，线性攻击者；

（c）指数防御者，指数攻击者；

（d）指数军备竞赛。

11.3 在练习 11.1 和练习 11.2 中，为攻击者和防御者两者考虑所有可能的随机、线性和指数策略的组合。从防御者的角度考虑，哪个是最差的组合策略？

11.4 对于图 11-1b 的杠铃故障树，说明 $r_k = V_R C_R + V_k C_k + V_L C_L$。

11.5 对于由图 11-1 和表 11-1 定义的简单网络，线性分配 $B = 10$，总结如表 11-2 所示，那么网络风险函数 Φ 的值为多少？

11.6 说明如果交换节点 4 和节点 5 上的分配，表 11-2 中的简单网络的网络风险会从 29 提高到 43 个单位。

11.7 将两条链路添加到图 11-1a 所示的简单网络中：一条从节点 4 到节点 2，第二条链路从节点 5 到节点 2。修改过的网络包含了多少条关键链路，具体有哪些？

11.8 如图 11-9 中所示，对于 $n = 200$、$p = 4\%$ 以及初始风险的预算范围为 10% 到 50% 的 2 - 规则小世界网络：构建一幅最大流量百分比与预算的对比图。在网络中的某些随机位置创建单个源和接收节点，并假定后果 C 和脆弱性减少费用 maxDA 为每次 100，如图 11-9 中所示。

11.9 Pointville 电网中攻击者和防御者预算等于 300 的例子中，使用军备竞赛算法，标准化网络风险为多少？对于指数攻击者分配策略和指数防御者分配策略时又为多少？有关秘密策略和公共策略之间的不同又如何解释？

11.10 对于以下网络，通过去渗流大约删除多少链路——表示成初始链路数的百分比，能造成图 11-8 的网络以概率 20% 分离：

（a）随机网络；

（b）无标度网络；

（c）小世界网络。

11.11 有必要在 Java 方法中计算 $\ln(\lambda_1)$ 和 $\ln(\lambda_2)$ 来检查用于零攻击者分配吗？为什么要或不要？

11.12 当使用防御者的预算为 20 单位、攻击者的预算为零单位和指数脆弱性缩减方程时，图 11-1a 和表 11-1 中的简单网络的优化防御者分配是什么？

第12章
Network Science: Theory and Applications

NetGain 网络

NetGain（纯收益）是一种包含了竞争性节点的网络属性，这些节点通常经过偏好连接或者与节点和链路关联的某些其他价值主张来调整网络结构以达到最优。NetGain 网络是网络化社会的专用模型，其中个体行为（节点）根据竞争者节点的价值主张、受欢迎程度或时尚流行而采纳一个产品、想法或服务。因此，在一类称为竞争者的节点与另一类被称为消费者的节点之间竞争，促成一个最终导致网络达到某一稳定固定点的涌现过程。在本章中，我们探索从一个随机的初始状态到一个非随机或同步固定点，演变成一个 NetGain 网络的各种方法。

自从20世纪50年代后期以来，NetGain 开始在很多文献中以多种名字出现，如增加收益、梅特卡夫定律[⊖]、网络经济、无摩擦经济和网络效应。所有这些术语都被用来描述任何大于其组成部件之和的网络化系统。NetGain 网络是通过根据某些内部或外部属性（如产品价格、经营效率或者消费喜好）来塑造网络拓扑的一种涌现形式。例如，一个随机的社会网络（音乐家的粉丝）因为很多节点喜欢将成为 hub 的某些节点（“歌星”）而可能演变成一个无标度网络（最著名的音乐家）。

我们认为一个 NetGain 网络的某种状态的涌现原因，可以追溯到一些网络或一些运行在微观层次的微规则。如果我们能标示这些驱动属性和微规则，那么我们还能解释 NetGain 系统是如何工作的，甚至还能解释它们是如何被驾驭的。本章我们重点放在经济市场，并探讨一个企业在网络经济中是如何运作、生存和蓬勃发展的？

我们假设 NetGain 网络准确地把消费者和竞争者建模为节点，把链路建模为产品需求，并把竞争者之间的竞争建模为一种涌现过程。我们研究当少数竞争者争夺消费者的注意力和购买力时会发生什么。这让我们从竞争者和产品数量的增加的角度来调查市场行为。消费者节点到竞争者节点的链路的数量是对需求的测量。竞争者根据它们的度（对产品的需求）来排名，消费者则通过在喜爱的竞争者节点上附加一条链路来指示它们对竞争者或其产品的忠诚度。

NetGain 的正式研究源自传染病学。在早期的通过市场的创新扩展模型中——所谓的扩散——将创新视为一种传染病。当我们假定人口均匀并且是静态的，那么早期的扩散模式就能很好地工作。但是，正如我们在本章中所见，纯粹的传染病模型已经不适合用来描述 NetGain 网络，特别不适合作为一种竞争结果经历网络拓扑改变的动态网络。

NetGain 不仅是传染病理论应用到通过静态人群的创新传播。NetGain 网络包含两种不同类型的节点——也就是消费者和竞争的生产者。此外，随着时间推移，竞争的持续，NetGain 网络在其度序列分布、节点和链路价值方面或者两者同时发生转变。NetGain 涌现重塑了网络，因此建模成动态的双向网络，$G(t) = \{P(t), C(t), E(t), f(t)\}$，其中 $P(t)$ 是生产者（竞争者）的集合，$C(t)$ 是消费者的集合，$E(t)$ 是连接消费者和竞争者的链路的集合，$f(t)$ 则是映射函数 $f: C \to P$。网络 G 是有向的，所以从消费者节点 c_i 到生产者节点 p_j 的链路表示为 $e: c_i \to p_j$，意味着 c_i 采

⊖ 网络的有用性与网络大小 n 的平方成正比。

用了 p_j 生产的一种产品、思想或者服务。

通过扩散转变的 NetGain 网络定义成产品、想法和服务在社会网络中的采纳过程。例如有关产品信息对个体的透露，个体按照通过影响链路获得的产品信息来决定是否要购买（或采用）或拒绝产品。但是，与先前研究的影响网络中的个体不同，当通过添加或删除一条到生产者节点的链路表示个体采用或丢弃一个产品时，NetGain 个体更改网络的拓扑。简单地说，扩散是在 NetGain 网络中添加或删除链路的过程。

扩散研究有关扩散过程的建模。已经设计了许多数学模型用来描述扩散过程。本章没有讲述那么多内容，我们仅介绍了基本的扩散要素，因为它涉及对社会网络的分析。对于更一般的讨论，我们推荐阅读 Stoneman 和 Forman 的著作（Stoneman, 2002; Forman, 2005）。在本章中：

1. 我们研究不同类型的同步并到达一个固定点市场份额的 NetGain 网络（包括简单垄断、多产品、新兴市场、创造性破坏和企业并购），定义为：market_share(v) = degree(v)/m，其中所有竞争者节点为 $v = 1,2,3,\cdots,n_p$。

2. 在一个简单垄断网络中的竞争结果是单一的垄断竞争者的兴起，假设一个随机的 NetGain 网络最初的链路数 m 等于消费者的数量 n_c，并由提供同等价值单一产品的竞争者来定义市场。

3. 竞争者在一个多产品 NetGain 网络中获得的固定点市场份额由一系列幂律分布来近似，当 $m \gg n_c$ 并且产品具有同等的消费价值，它定义一个垄断竞争者的市场地位。因此，多产品市场实质上就是垄断市场。

4. 创新者就是首次向市场推广某个新产品、服务或者想法的一个竞争者。创新者都是第一个倡议者。一个模仿者就是一个复制或者模仿创新者的竞争者。模仿者也可以成为快速追随者，尤其是如果他们快速复制创新者时更是如此。一切都是平等的，一个创新者就是因为第一个推向市场而简单地上升到垄断地位。

5. 在新兴市场网络中的固定点市场份额，主要是由消费者按泊松过程选择（随机地）从多个竞争者中订购产品的概率来控制的。这就在当 $n_p \geq 3$ 个竞争者时，在 ｛63%，26%，8%，2%］的固定点市场份额分布上有了一个下限。数字 3 很重要，这表明不管进入市场的竞争者有多少，新兴市场只能支持 2 至 3 名领头的竞争者。

6. 市场的速度定义成，市场速度 =（新链路到达的速率/新竞争者到达的速率）。随着市场速度的增加，创新者（第一个倡议者）拥有超越模仿者（快速追随者）的优势。反之，市场越慢，模仿者就会有更好的超越创造者的机会。这也许可以解释为什么倡导者会在高科技产业具有优势的原因。

7. Schumpeter（熊彼特）的创造性破坏是一种由创造性破坏网络建模的涌现进程，这个网络将消费者的价值主张表示成竞争者节点的值（Schumpeter, 1942），即弱竞争者增加其节点值而强竞争者增加其利润率（更低的节点值）。这导致了垄断竞争者的倒台和随后的领导者的出现，这个过程称为创造性破坏。我们证明创造性破坏会导致产业生命周期——市场领导者的盛衰。

8. 在随机创造性破坏网络中的交换平方根律仿真表明，一个稳定的固定点涌现，与具有多少竞争者 $n_p \geq 3$ 开始主导无关。此外，一个竞争者将会涌现并继续处于主导地位。如果最小竞争者的阈值足够小，也会涌现一个有较少份额的快速追随者，并一直处于第二。最小竞争者的阈值越低，快速追随者涌现的数量就会越高。在创造性破坏网络中垄断阈值 $m = 60\%$ 时，我们说明了领导者的固定点市场份额大约是：market_share_leader(m,p) = $m + 5 * p * (1 - p - m)$，其中 m 是定义垄断的市场份额，并且 $0 \leq p \leq (1 - m)$。

9. 在企业并购网络中的竞争者兼并是指两个竞争者节点的兼并，因此较小市场份额节点的链路被指定给较大市场份额节点。我们证明了企业并购在与创造性破坏网络联合时必然会加快垄断或双寡头形式的形成。我们进一步证明，在一个创造性网络中，领导者的固定点市场份额加

上兼并份额（垄断阈值 $m = 72\%$）是：market_share_leader(m,p) = $m + 6.5 * p * (1 - p - m)$，其中 m 是定义垄断的市场份额，并且 $0 \leqslant p \leqslant (1 - m)$。

12.1 经典扩散方程

社会网络为产品、思想或服务通过社会扩散传播提供了一个向量。产品、思想和服务一般被分为创新和模仿。*创新*是任何产品、服务或思想在其类别中首次出现，而*模仿*是一个创新的变异。我们将扩展采纳过程称为工艺创新或模仿*扩散*。

从商业的角度来看，*扩散*是产品采纳的过程，它在 NetGain 网络中仿真成消费者节点和产品节点之间插入的链路。*采纳*就是一个个体决定使用一种新产品、思想或服务——基于采用前面章节描述的各种机制提供给消费者的信息。例如，当人们谈论产品时，口碑扩散传播便会发生，当广告出现在电视屏幕上时，媒介传播便会发生。

为了理解扩散机制，我们将采纳过程建模成涌现网络中的微规则，然后按照网络拓扑分类结果宏观尺度模式。例如，我们根据简单的产品认可测量显示创新的扩散，例如在无标度网络拓扑中的流行结果。此外，我们在社会网络中观察到作为竞争者节点的状态或价值主张结果的宏观尺度模式的形成。收敛是由于节点值围绕一个稳定的固定点来同步社会网络。例如，当将一个平方根微规则反复应用到 NetGain 网络的节点和链路时，它经常以市场份额的形式收敛到一个固定点。

竞争者节点中统治*双寡头*的涌现，是初始的随机网络中众多节点竞争的结果。从本章仿真中涌现的持续垄断和双寡头格局，产生了有关主导竞争者的必然性的非常有趣的问题。在任何公开竞争的市场中是一个、两个、还是三个公司主导，这是必然的吗？难道小企业总是不可避免地消亡吗？

12.1.1 市场扩散方程

当然，没有必要使用网络科学来建模扩散。事实上，最早的扩散数学模型是基于数学上的流行病方程[⊖]。这些方程假设创新或模仿的扩散如同传染病的传播一样。产品越有感染力，创新的扩散就会越快。从产品市场营销来讲，受感染的个体被称为*市场渗透*和*市场份额*，而不是*感染密度*。利用*采纳率*代替*感染率*描述创新传播的速度。

高市场份额（即在前一时间步中已经采用新产品的人口百分比较高）增加了采纳率，反过来又提高了市场渗透率等。这种反馈机制也是*偏好连接*的另外一种解释。在一个理想化的世界中，偏好连接导致了类似于第8章中获得的逻辑斯蒂增长曲线（Ryan，1943）。

在 NetGain 网络中，描述采纳率与时间之间的函数可能会有所不同，因为网络的拓扑可能会抑制或加速采纳率。这里有一个我们要提出的问题，“网络扩散受拓扑影响吗?”如果答案是“否”，那么，从流行病学中得到的经过检验且可靠的方程就足够了。

在时间 t 的市场份额，$0 \leqslant s(t) \leqslant 1$，就像社会网络中的*感染密度*一样，是产品扩散中有效性的一个基本测量。它是已经采用产品的人口的一个分数。很明显，市场渗透率随时间变化，初始值接近零然后提高到了最大，然后就维持在那里，或者经过一段时间后下降。*采纳率*是市场份额随时间变化的一阶导数：$\delta s/\delta t$。根据市场份额与时间（$s(t)$ 与 t）来绘图，给出了产品的*采纳曲线*。

图 12-1 说明了一些著名的常见产品的采纳曲线，并建议 $s(t)$ 就是逻辑斯蒂函数。我们在第8章中学习过 Kermack-McKendrick 方程的最基本解是一个逻辑斯蒂曲线。这似乎证实了扩散和传

⊖ 即使在今天，术语“病毒式营销”仍被用来指代使用传染病建模描述创新的扩散。

染扩散的等价性。事实上，Bass 是第一个应用流行病模型来解释产品采用的（Bass，1969）。图 12-1 证实了 Bass 模型。

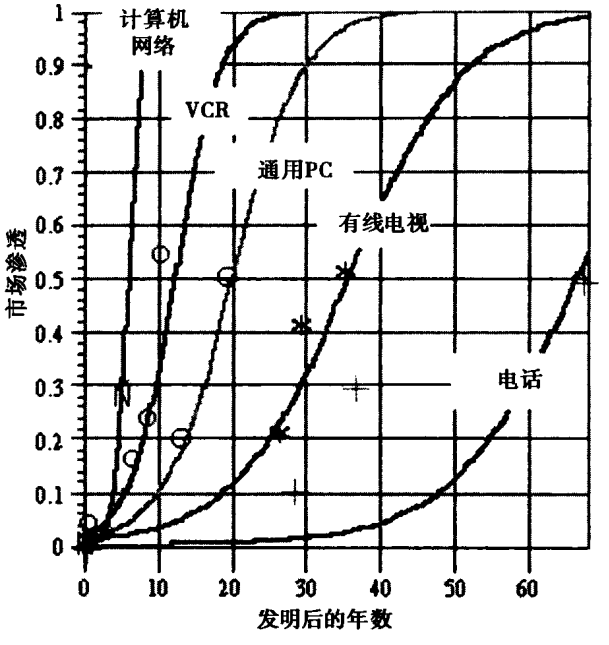


图 12-1 新创新通过社会的扩散：市场渗透表示为自产品发明以来采纳产品的人数占总人数的分数

Bass 方程

$$s(t) = \frac{1 - e^{-bt}}{1 + ae^{-bt}}$$

$$s(0) = 0$$

$$t > 0$$

（其中 a 和 b 是常量参数）。在图 12-1 中采用的 Bass 函数可以通过适当的最小二乘近似估计参数 a 和 b。显然，电话、有线电视、个人计算机的采纳率表明，互联网的市场渗透率会稳定地随着技术的进步而增长。在过去的 100 年里，达到 50% 渗透率的时间长度已经下降了 90% 以上。因此，参数 a 和 b 会因创新不同而不同，并随时间而改变。

在 Bass 方程中，由不同的 a 和 b 参数值产生呈指数地从零上升到一个拐点 $t^* = \ln(a)/b$ 的 S 状采纳曲线。然后，曲线逐渐变平，直至达到饱和点 s^* 为止，在图 12-1 中为 100%。我们可能会将指数上升的部分看成代表收益增加的阶段，而后半部代表收益递减的阶段。拐点 t^* 表示一个从早期采纳者到一个成熟市场的转折点。Fisher 和 Pry（Bass，2001；Norton，1987）提出了一个简化而密切相关的扩散模型。在他们的模型中，市场份额的时间导数 $\delta s/\delta t$ 和已经采用的市场份额 $s(t)$ 以及仍然存在的份额 $[1 - s(t)]$ 成正比：

Fisher-Pry 方程

$$\frac{\partial s(t)}{\partial t} = rs(t)[1 - s(t)]$$

$$s(0) = 0$$

$$t > 0$$

（在这里 r 是一个速率参数）。如果 $s(0) = 1/2$，逻辑斯蒂方程能像前面一样求解：

$$s(t) = \frac{1}{1 + e^{-rt}}$$

$$t > 0$$

其中 r 是直线的斜率：

$$\ln\left(\frac{s}{1-s}\right)$$

Fisher-Pry 模型与 Bass 模型有关，但两者又有不同。它们不仅有不同的初始条件，而且 Fisher-Pry 公式会产生不同斜率的逻辑斯蒂曲线[⊖]。我们可以比较两种模型，通过转换 Fisher-Pry 时间比例因子用 $s(0) = \frac{1}{2}$ 代替 $s(0) = 0$ 来说明在初始条件下的差异。用 $(t - t^*)$ 替代 t，并插入一个偏差 y 来产生一个 Bass 方程的近似。以 Bass 参数 a、b 和 t^* 表示 Fisher-Pry 扩散方程如下：

修改过的 Fisher-pry 方程

$$s(t) = \frac{1}{1 + e^{-r(t-t^*)}} + y$$

$$s(0) \sim 0$$

$$t^* = \frac{\ln(a)}{b}$$

$$y = \frac{1}{1 + e^{bt^*}}$$

上述先驱性的工作假定均匀混合的市场人群，并且忽视了社会网络的拓扑结构。它不足以解释某种网络扩散是如何工作的。具体来讲，它无法预测什么样的网络拓扑会从一个涉及许多竞争者的市场中涌现。我们在 NetGain 网络模型中解决该问题，这个模型表明偏好连接就能导致类无标度的拓扑。这证实了 Bass 和 Fisher-Pry 模型适合于简单、单一产品市场的扩散，但它未能解决多竞争者、多产品市场问题。此外，NetGain 模型可能掌握着用以解释意想不到的颠覆性技术如何取代市场领导的关键，这与 Schumpeter 的创造性破坏理论相一致（Schumpeter，1942）[⊜]。

我们通过向 NetGain 网络中添加一个涌现组件来扩展扩散理论：竞争者的节点经过反复应用微规则来争夺客户链路。当客户比较产品时，通过一系列微规则模拟客户使用的价值主张的应用。允许竞争者增加或减少其产品的价值，以吸引顾客或最大化利润。

我们证明了一个可变的价值主张导致了带有一个主导竞争者和一个（或两个）不那么成功的追随者的网络的稳定状态。这个理论认为，包含一个垄断和一个从属竞争者的网络，对某类网络而言是最稳定的状态。这个主张可以由下面描述的模拟实验得到证实。

12.1.2 简单 NetGain 网络

在一个有限维数、按照 Bass 和 Fisher-Pry 扩散理论表现的随机网络中，市场份额是否会扩展？我们进行实验以寻找答案。考虑下面的 NetGain 网络。一个随机连接的有向网络带有 n_c 个消费节点、n_p 个竞争者节点和 m 条有向链路，最初是按如下组织的，即从消费者 v_c 到竞争者 v_p 的一条链路意味着消费者 v_c 是一个竞争者 v_p 的消费者。网络最初是随机的，因为每一条链路连接到一个随机选择的竞争者节点上。我们允许链路数比消费者数量更多，以便可以建模多产品公司。

图 12-2 显示了由三个竞争者和 40 个消费者组成的网络。程序 NetGain.jar 根据 NetGain 节点功能标记颜色代码：消费者是纯黑色圆点，竞争者是带有节点索引的矩形框，而在矩形内标注节点市场份额或产品价值。市场份额不能超过 100%，其非常简单的计算如下：

⊖ Fisher-Pry 逻辑斯蒂曲线稍低于同等的 Bass 逻辑斯蒂曲线。

⊜ 奥地利经济学家和政治科学家，Joseph Schumpeter（1883—1950）认为资本主义导致“创造性破坏”，其中现任主导者被破坏，从而被新的市场领导者所替代。

$$\text{Market\_share}(v_p) = \text{degree}(v_p)/m; \text{其中 } m = \text{链路数}$$

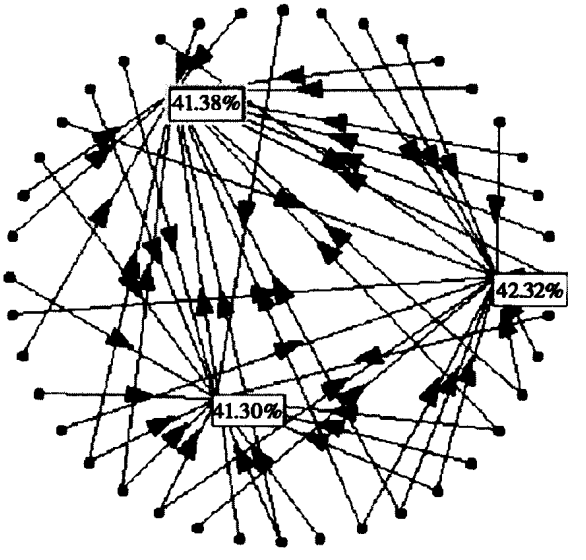


图 12-2 具有 $n_c = 40$ 个消费者、$n_p = 3$ 个竞争者、$m = 50$ 个随机链路的随机 NetGain 网络。对竞争者节点进行编号，并标记其市场份额百分比（如图所示）或价值主张

事实上，我们不允许消费者和竞争者节点之间的重复链路，因此当 $m > n_c$ 时，市场份额将低于 $\text{degree}(v_p)/m$。在这种情况下：

$$\text{Market\_share}(v_p) \leqslant \frac{n_c}{m}$$

竞争节点包含一个值，这可以解释成从市场角度来看的产品价值、服务或观念。例如，商品的价值可能是由于其相对较低的价格、高质量或时尚所决定的。产品价值是从购买者的角度来看，是当他们在竞争者之间比较产品时来判断的。这就是所谓的每一个竞争者与客户沟通的价值主张。此时，我们先假设对所有产品和竞争者而言，价值是相同的。

链路是从消费者指向竞争者节点，表明消费者订购由竞争者提供的产品。多条（但不重复）链路连接到消费者的节点可以解释成分裂的忠诚，或代表来自不同公司的不同产品。如果有比消费者更多的链路 $m > n_c$，我们就假设 NetGain 网络代表一个多产品市场。

随着时间的推移，当消费者改变偏好和忠诚时，随机 NetGain 网络会发生什么？由什么决定客户在如图 12-2 所示的市场中的偏好？我们通过进行如下“人气竞赛”涌现过程来解决这些问题。在每一个时间步，随机选择一个竞争者节点 $V_{\text{challenger}}$ 和一个指向节点 $V_{\text{challenger}}$ 的随机链路。通过将链路连接到最高度（市场份额）节点来重联链路。这将模拟人类群体行为——人们希望拥有别人所拥有的。这亦是第 6 章描述的无标度网络生成过程的简单变化。

多产品涌现

1. 最初生成随机 NetGain 网络，带有 n_c 个消费者节点，n_p 个竞争者节点，m 条随机放置的将消费者连接到竞争者节点的有向链路。

2. 无限地重复，或者直至用户停止为止：

a. 随机选择 any_link 链路，定义 any_link 将 consumer_node 连接到 incumbent_node。

b. 随机选择竞争者节点 challenger_node。

c. 如果 challenger_node 的度超过 incumbent_node 的度就交换 any_link 的头部，从 incumbent_node 到 challenger_node。否则不做任何事。

从这种简单的过程会导致什么样的涌现网络？直观上，随机网络变换成星形网络，即拥有最

大初始市场份额的竞争者将增加其市场份额，因为链路被吸引到具有最大度的节点上。事实上，这种直觉只适用于 $m \leqslant n_c$ 情况下。当消费需求缺乏时（有限的链路数），最大市场份额的竞争者会得到更多的市场份额，直至它成为垄断（即所有 100% 链路都指向它）为止。

因为竞争者的度降为零——他们是被市场所淘汰的牺牲品，他们就会退出市场。在每一个市场都有一些落伍者，可能有少量参与者（既不是占主导地位，也不是落伍者的竞争者），也可能会有一个垄断竞争者。在这种情况下，所有的竞争者都被淘汰掉，只留下一个垄断竞争者。

多产品涌现过程将 NetGain 网络转换为由市场份额百分比定义的同步稳定状态，一旦到达其固定点就永远不会再改变了。我们将此称为稳定的融合网络和不变市场份额百分比分布，一个汇聚网络的固定点。在本章中学习的所有 NetGain 网络都是同步的，并在有限时间内到达固定点市场份额分布。

简单单一产品网络证明并支持 Bass 和 Fisher-Pry 产品扩散模型。与初始状态无关，单一产品网络会达到垄断固定点。领先的竞争者根据 Bass 和 Fisher-Pry 模型，达到固定点市场份额（参见图 12-3）。这可以通过运行程序 NetGain. jar 观察到，随着时间的变化网络不断演变，市场份额数据拟合逻辑斯蒂方程。显然，当 $m \leqslant n_c$，有稍多连接的竞争者就会按照 Bass 方程达到顶峰，它就变成垄断。

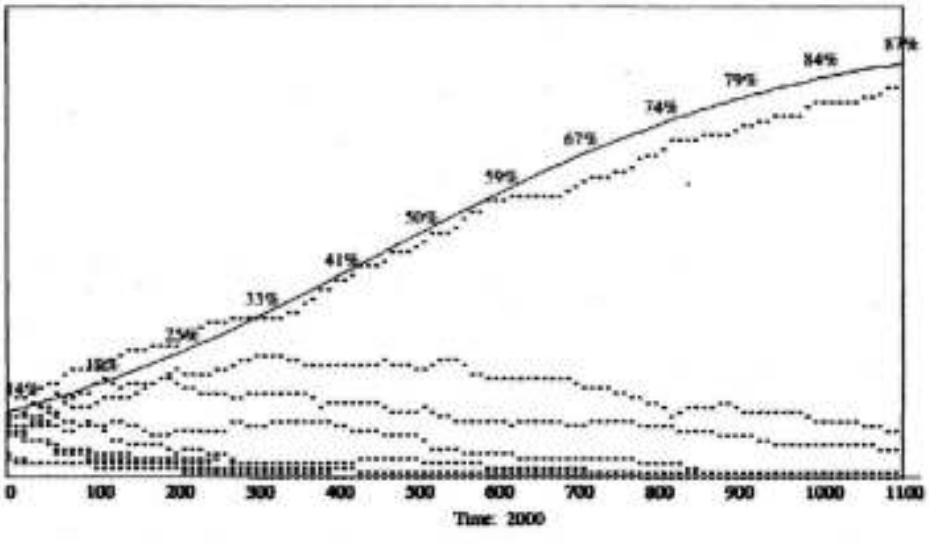


图 12-3 初始状态为 100 个消费者、5 个竞争者、100 条链路的随机网络的市场份额，证实了 Bass 方程

因为 NetGain 网络的初始状态是随机的，竞争者最初分配几乎相同数量的链路。但是，一个幸运的竞争者意外地获得比别人更多的链路，而使它走向垄断。市场份额会按照 Bass 方程上升：

$$s(t) = \frac{1 - e^{-a}}{1 + ae^{-bt}}; \qquad 0 \leqslant t \leqslant 1000$$

其中 $a = 5.86$，$b = 0.0039$。这个方程非常完美地匹配了图 12-3 的市场份额数据！其他四个竞争者的市场份额在早期阶段也会上升，达到顶峰后会随着市场领导者加速而下降，这是由于偏好连接提高了回报效应。竞争者拥有的市场份额越多，得到的也就越多。

12.2 多产品网络

更加有趣的、现实的网络包含提供多种产品的多个竞争者，使得 $m \gg n_c$。每个消费者都有多种产品的偏好，这可能扩散到多个竞争者。事实上，消费者将忠诚于多个竞争者就是因为有比消费者更多的链路。由于大量的链路，多产品网络将演变成为市场领导者的寡头垄断。

我们以同样的方式模拟这种市场，从一个随机的拓扑开始，随后反复地应用偏好连接。再次，一个主导竞争者涌现，但是也包括部分参与者和落伍者的涌现。特别是，多产品 NetGain 仿真结果显示了很多竞争者与主导者甚至垄断竞争者共存。相对于上面检查过的单一垄断的简单涌现，多产品市场支持多个竞争者。固定点市场份额分布支持参与者、落伍者和一个主导竞争者的共存。

图 12-4 画出了 NetGain 网络中五个竞争者的仿真结果，其中 $n_c = 100$ 个消费者，$m \geq n_c$ 条链路（需求）。竞争者按照市场份额排名，$s_1 \geq s_2 \geq s_3 \geq s_4 \geq s_5$。直观上，我们期望前 100 条链路分配给 s_1，接下来的 100 条链路分配给 s_2 等，直到所有的链路都被分配给了竞争者为止。但是，这种情况不会发生。相反，某些链路，一旦被分配给某个竞争者，就永远不会离开。随着时间的推移，NetGain 网络达到一个与表象相反的固定点。不再是单个占据主导的竞争者涌现，而是寡头竞争者的涌现！为什么会这样？

在图 12-4 中，领先竞争者 s_1 的市场份额曲线很容易确定，因为主要竞争者不断得到更多的市场份额，直到达到由消费者数量决定的限制为止。但为何其市场份额随着消费者链路比例上升而下降呢？采纳曲线显然不是一个 Bass 或 Fisher-Pry 逻辑斯蒂曲线。

随着链路总数增加超过 n_c，以及领导者达到它的 100 条链路的最大值，比例 $100/m$ 会按照以下关系下降：

$$\text{Market\_share}(s_1) = \frac{n_c}{m}; n_c \leq 100 = \frac{100}{m}; n_c > 100$$

因此，主导竞争者的市场份额随着链路数量的增加反而减少。图 12-4 中的曲线显然是 $O(1/m)$，这是一种幂律分布。

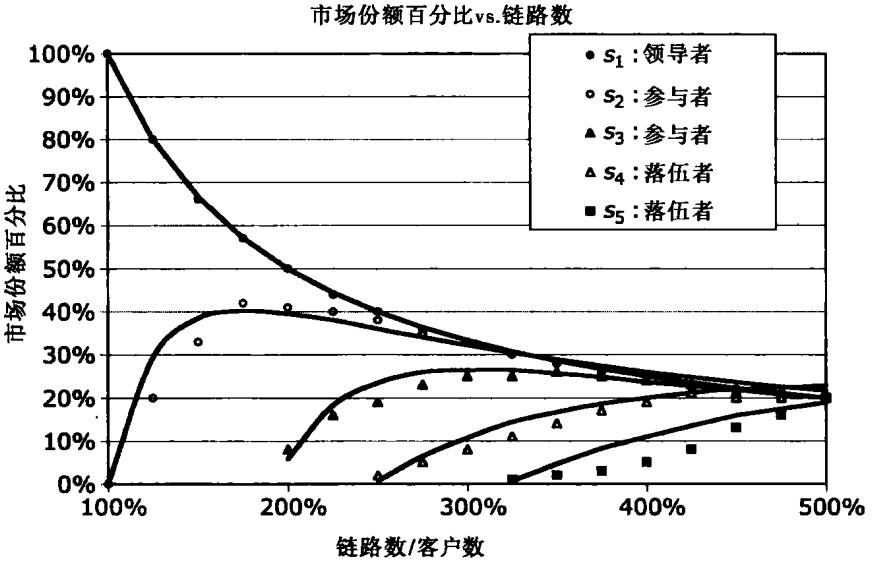


图 12-4　$n_c = 100$ 个消费者、$100 \leq m \leq 500$ 条链路、$n_p = 5$ 个竞争者的最初涌现结果。对 m/n_c 的低值而言，落伍者被淘汰。对于高值而言，几个竞争者节点成为参与者

在一般情况下，多产品的涌现和无标度网络涌现紧密相关，图 12-4 中所有的数据应该拟合幂律分布。幂律分布 $1/[n_c/m]$ 非常好地拟合于 market_share(s_1)，但是它适合估算其他竞争者的市场份额吗？答案是肯定的，由此产生的修改过的幂律分布取代单一产品 Bass 和 Fisher-Pry 扩散模型。

当竞争者 s_1，s_2，…，s_i 达到其最终的市场份额值时，设 $A_i = \sum_{j=1}^{i} \text{market\_share}(s_j)$ 为消耗的部分市场份额，当寡头涌现发生时，设 B_i 和 C_i 代表从不愿转换链路而导致的粘滞性。我们通过以下幂律分布与图12-4中显示的数据的曲线拟合来获得 B_i 和 C_i。然后，竞争者 $s_{i+1}(i \geq 1)$ 的市场份额近似如下：

$$s_{i+1} = \frac{m/n_c - B_i - A_i}{(m/n_c - C_i)^{q_i}}; i \geq 1$$

$$A_i = \sum_{j=1}^{i} \text{market\_share}(s_i)$$

其中 B_i、C_i 和 q_i 是通过对图12-4中的数据的曲线拟合获得的。

表12.1总结了通过图12-4中的每一个竞争者的曲线拟合获得的近似参数。随着链路数量（需求）增加，A_i 和 B_i 也由于链路的粘滞性而增长。随着链路数量的增长，所有竞争者达到一个同等的固定点（图12-4中每个20%），因为对于所有竞争者有足够的链路以便达到其最大值100条链路。

直观上，占主导地位的竞争者应该获得100条链路，那么下一个占主导地位的竞争者应该获得接下来的100条链路，依此类推，直至连接了所有的链路为止。例如，如果 $m = 300$，我们希望 $s_1 = 33\%$、$s_2 = 33\%$、$s_3 = 33\%$，所有其他竞争者的份额下降到零。反之，反复仿真图12-4中的网络就得到 $s_1 = 33\%$、$s_2 = 33\%$、$s_3 = 25\%$、$s_4 = 8\%$、$s_5 = 0\%$，三个落伍者分享最后的33%！为什么？

表12-1 图12-4的粘滞性参数

竞争者 i	B_i	C_i	q_i	m/n_c 的范围
1	0	0	1	$m \geq n_c$
2	0	0	1.95	$m \geq n_c$
3	1.0	0.75	2.0	$m \geq 2n_c$
4	1.5	0.10	1.6	$m \geq 2.5n_c$
5	2.25	0	1.45	$m \geq 3n_c$

多产品网络汇聚到一个寡头垄断和固定点市场份额违反直觉，因为一些消费者被链接到所有的竞争者。当对消费者节点 v_1 和 v_2 发生这种情况时，因为他们已经连接了，因此就不能将链路换到一个更强的竞争者。不允许重复的链路，因此偏好连接也不再具有操作性。

随着链路数量的上升，消费者节点连接到一个更强的竞争者的概率也会上升。粘滞性不仅是一个消费者对竞争者的忠诚度——它也是一个统计事实，即随机选择的消费者可以具有 n_p 条链路。事实上，将一个消费者链接到 k 个竞争者是一个由二项式分布控制的泊松过程。在多产品网络中寻找期望的任何竞争者的固定点市场份额留给读者作为练习。

多产品市场固定点近似方程的应用很直截了当。只需从表12-1中获取参数值并带入到方程即可。例如，当 $m = 250$、$n_c = 100$ 时，五个竞争者中每一个的近似市场份额是多少？运用导出的幂律方程，我们得到：

$$m/n_c = \frac{250}{100} = 2.5 \quad \text{所以} \quad \frac{1}{[m/n_c]} = \frac{100}{250} = 0.40$$

$$s_1 = 100/250 = 40\%$$

$$A_2 = 40\%$$

$$s_2 = \frac{[12.5 - 0 - 0.4]}{[(2.5 - 0)^{1.95}]} = \frac{2.1}{5.97} = 35\%$$

$$A_3 = 40\% + 35\% = 75\%$$

$$s_3 = \frac{[2.5 - 1.0 - 7.5]}{[2.5 - 0.75]^{2.0}} = \frac{0.5}{3.063} = 16\%$$

$$A_4 = 91\%$$

$$s_4 = \frac{[2.5 - 1.0 - 0.91]}{[(2.5 - 0.1)^{1.6}]} = \frac{0.09}{4.06} = 2\%$$

$$A_5 = 93\%$$

$$s_5 = 0 \quad 因为 \quad m < 3n_c$$

近似值 s = [40%，35%，16%，2%，0%] 有利于与图 12-4 中的数据进行对比，通过仿真产生 [40%，38%，19%，2%，0%]。不考虑粘滞性估计固定市场份额得到 [40%，40%，20%，0%，0%]。

多产品 NetGain 网络演变成无标度网络。到节点的链路的准确分布依赖于总需求（m 条链路），但总固定点分布是急剧下降的幂律分布。多产品涌现导致寡头垄断而不是垄断，由少数领导人、一些参与者，很可能还有一个落伍者描述。

我们已经证明一个市场的领导者在简单的单一产品的网络中通过偏好连接涌现，并遵循 Bass 扩散方程。从这个方面考虑，有限的、非均匀的 NetGain 网络行为表现由前面的扩散理论所预测。但在这个过程中要进行一些假设：（1）所有产品被认为是消费者所谓的同等价值；（2）网络是从一个充分展开但随机连接的 NetGain 网络开始的；（3）没有足够多的链路进行循环，所以只有一个竞争者存活。

在一个多产品网络中，链路比消费者多，所以出现寡头垄断而非垄断。产品仍然被消费者认为是同等价值，网络随机启动，但有一个“剩余”的链路。这使得许多竞争者生存下来，导致寡头垄断。但竞争者受 100 条链路限制，并且链路粘滞性导致固定点份额服从幂律分布。粘滞性实际上是一个生物的统计事实，链路根据泊松过程被分配给节点，该过程允许一些消费者节点连接到所有的竞争者。这剥夺了偏好连接的影响。

这个实验的结论之一是，在经济落后的社会网络中垄断出现并蓬勃发展，即稀疏 NetGain 网络演变为无标度垄断网络。相反，富裕的社会网络即密集网络转变为寡头垄断。这些都是类无标度网络，而不是纯粹的无标度网络。小康社会能够支持许多生产者，但是落后的社会却不可能。

12.3 NetGain 网络涌现的 Java 方法

NW_doCompete( ) 方法实现网络中的多产品涌现过程，其中网络中的消费者节点 n_c = nConsumers，竞争者节点 n_p = nCompetitors，链路 m = nInputLinks。Java 实现非常完美地解释了之前描述的多产品涌现算法。

首先，随机选定的 any_link：consumer → incumbent 是作为候选链路而被重联的。接下来，选择候选挑战者节点，其度与现任者节点的度进行比较。如果挑战者的度比现任者的度大，那么链路 any_ link 可能被切换到 consumer → challenger。

Java 方法非常保守。它检查可能的被零除的运算，它使用方法 NW_doAddLink( ) 来检查可能重复的链路。重联链路在老的链路被删除之前通过方法 NW_doCutLink( ) 插入。全局字符串 Message 向用户传达竞争状态，通过一种没有在这里提过的方法向用户显示。由 NW_doCompete( ) 使用的其他全局性参数是 nInputConsumers、nInputCompetitors、nInputLinks、nConsumers 和 nCompetitors。这些参数是自定义的，能够通过 NetGain. jar 程序的偏爱对话框来加以改变：

```
public void NW_doCompete(int t) {
    if(nInputConsumers <= 0 || nInputCompetitors <= 0 || nInputLinks <= 0) return;
    int any_link = (int)(nLinks*Math.random());          //Random link
    Link e = Links[any_link];
    int consumer_node = e.from;           //consumer node -> competitor node
    Node incumbent = nodes[e.to];       //links always point to competitor
    int challenger_node = nConsumers+(int)(nCompetitors*Math.random());
    Node challenger = nodes[challenger_node];
    double denominator = challenger.degree;
    if(incumbent.degree <= 0                    //Eliminate competitors with no share
          ||
       denominator <= 0) return;              //Eliminate divide-by-zero error
    if(challenger.degree > incumbent.degree) {  //Switch
       if(NW_doAddLink(consumer_node, challenger_node)) {//Avoid duplicate link
          NW_doCutLink(any_link);
          Message = "Switching to challenger: "+
                "[degree(challenger)="+Long.toString(challenger.degree)+
                ">"+Long.toString(incumbent.degree)+" = degree(incumbent)]";
       }
    }
 }//NW_doCompete
```

12.4 新兴市场网络

一个*新兴的市场*是一个正在发展中的市场——客户的链路数从零开始，随着时间的推移而不断增加。随着链路数的增加，新的竞争者节点不断出现，以解决由链路增长表示的日益增长的需求。通常，一个创新推出新产品，并享有一段时间垄断。这种创新者被称为*倡导者*。其他的竞争者很快加入，尤其是如果第一个倡导者赚取了利润更是如此。这些模仿者出于显而易见的原因而被称为*快速追随者*。

当倡导者和快速追随者竞争对市场刺激使需求增加时，更多的竞争者急于进入市场，直到市场充满了竞争者为止。但随后的需求高峰和链路数保持不变。在这一点上新兴的市场成为一个*成熟的市场*，并且竞争者的数目因为竞争开始下降。这被称为*淘汰阶段*，因为不太成功的竞争者失去市场份额而破产。最后只有极少数竞争者保留下来。

全新的产品类型在其采用的前几年里创造新兴市场。盒式磁带录像机 VCR 是一个经典的例子——在引进第一批 VCR（由 RCA）之前没有 VCR 市场。它用了 12 年时间使得 50% 的美国家庭拥有 VCR（参见图 12-1）。最终，VCR 市场已经达到了一个高峰，而随后的市场淘汰只留下一小部分竞争者。在 VCR 例子中，一个新的创新——DVR（数字录像机）取代它，这个过程又重新开始。

我们想要理解新生市场的动态特性。那么它们是否遵守 Fisher-Pry 采纳模型呢？它们是否总是造成垄断、双寡头垄断或寡头垄断？新兴市场的性质是什么？对一个希望成为主导者的竞争者来说，最好的策略是什么？我们通过涌现 NetGain 网络仿真来寻找这些答案。

12.4.1 新生市场的涌现

考虑一个新生市场最初是由单一竞争者即创新者所垄断。以额外链路的形式表现的需求假设以常数速率稳定地增长。例如，每隔五个时间步添加一个新的客户链路。同样地，因为需求的增加，模仿者被吸引到新兴市场上来。通过每隔 20 个时间步添加一个新的竞争者节点来模仿竞争者项，直到添加了 n_p 个竞争者为止。比率 20/5 =4 是对新兴市场*速度*的测量，因为每当一个竞争者被添加时会添加四条链路（需求），最多高达 n_p 个竞争者。需求继续增加到极限 m = nInputLinks。

当将链路和节点添加到网络时，已有的链路按照偏好连接重联。市场份额被定义成节点度除以链路总数，因此随着总数上升，可以推测出竞争者会慢慢地失去市场份额，除非它从较弱的

竞争者获取远超过它的份额。因此，两个因素决定了最终的竞争者的固定点份额，即新链路的添加和偏好连接。

假设经过 nNewLinkInterval 时间步后定期添加一条新的链路，每经过 nNew Node Interval 时间步后添加一个新的竞争者节点。进一步假设随后市场份额的增长是通过偏好连接发生的。下面取自程序 NetGain. jar 的代码片段简明扼要地反映了一个新生 NetGain 网络的增长，而方法 NW_doCompete( )描述前面模拟过的偏好连接：

```
int consumer_node = (int)(nConsumers*Math.random());
int challenger_node = nConsumers+(int)(nCompetitors*Math.random());
//Add competitor
if(doNascent && nCompetitors < nInputCompetitors
            &&
     t == nNewNodeInterval*(t/nNewNodeInterval)){
            NW_doAddNode(Long.toString(nNodes));
            nodes[nNodes-1].color = Color.blue;
            nCompetitors++;
            NW_doAddLink(consumer_node, nNodes-1);
            Message = "New Competitor:";
    }
//Add new link
if(doNascent && nLinks < nInputLinks && challenger_node >= nConsumers
            &&
       t == nNewLinkInterval*(t/nNewLinkInterval)){
                if(NW_doAddLink(consumer_node, challenger_node))
                   Message = "New Customer:";
    }
```

请注意，一个新的竞争者也有一条新的链路连接到它，因此每一个新的竞争者最初至少有一个消费者。竞争者的节点在程序 NetGain. jar 中都被显示成蓝色以区别于消费者。只要 n_p 个竞争者以固定间隔添加，就不再有其他添加。

新的链路也是以固定时间间隔添加，但是方法 NW_doAddLink( )检查是否重复，如果它不能插入一条链路，则返回 false。重复链路得以避免，这意味着 n_p 是任何可能的消费者节点最大的度。所有 nConsumers - 1 之上的竞争者节点被编号，这就是为什么该代码段要检查 challenger_node > = nConsumers 的原因。

12.4.2 新兴市场固定点

我们将新链路连接到现存的和新的竞争者上建模成*泊松过程*，这与创建一个随机网络生成程序是相同的。然而，并不是将消费者节点连接到其他消费者节点上，我们随机选取消费者节点连接到竞争者节点。将一条链路连接到 n_c 个消费者节点之一的概率是 $1/n_c$，将 k 条链路连接到某个消费者节点的 m 次尝试的概率也就是二项式分布：

$$B(m,k) = C(m,k)\left(\frac{1}{n_c}\right)^k\left(1-\frac{1}{n_c}\right)^{m-k}$$

直观上，我们期望市场份额会在所有 n_p 个竞争者之间平分，但因为是二项式分布，这种现象就不会发生，这允许消费者连接到多个竞争者的概率为 Prob（degree ≥ k）：

$$\text{Prob}(\text{degree} \geqslant k) = \sum_{j=k}^{m} B(m,j)$$

消费者节点至少有 k 条链路的概率，并且因此它的度为 k 或更大，显示在 m = 100 条链路、n_c = 100 个竞争者的图 12-5 中。这一统计事实的重要性对新兴市场的固定点是非常关键的，因为当某个消费者节点连接到所有的竞争者时，它可以阻止由于消费者偏好连接链路重联。与竞争者的度无关，当消费者已经连接到竞争者时，就不可能再有重联了。连接到所有竞争者的消费者不能再重联。

二项式分布的粘滞性，意味着新兴市场将在竞争者之间大致分成如下比例（参见图12-5）：

新兴市场的固定点市场份额：$n_p = 5$

市场领先者：63%

快速追随者：26%

落伍者：8%，2%

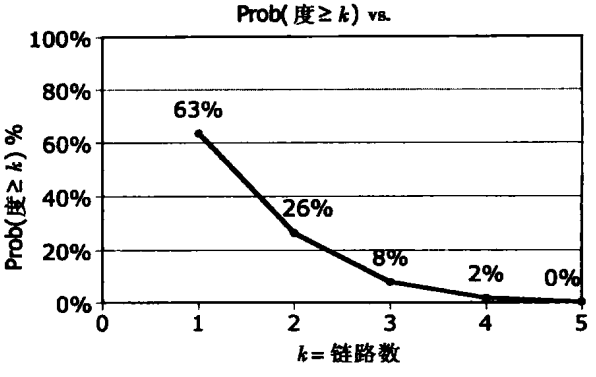


图12-5 消费者节点将至少有 k 条链路的概率与 k 关系图，其中 $m = 100$，$k = 1, 2, 3, 4, 5 = n_p$，$n_c = 100$ 个消费者

这是概率论的结果，而非竞争或偏好连接得出的结果。但是，当 $n_p \leq 3$ 时，偏好连接接管并且更多的主导竞争者“妆起”没有被落伍者消耗的链路。

新兴市场的固定点市场份额：$n_p = 4$

市场领先者：65%

快速追随者：27%

落伍者：8%

新兴市场的固定点市场份额：$n_p = 3$

市场领先者：73%

快速追随者：27%

落伍者：0%

市场的速度是新添加的链路数量与新添加的竞争者节点数量之比。程序NetGain.jar允许用户以时间间隔为单位输入这些比率。因此，举例来说，如果一条新的链路以10嘀嗒节拍的间隔添加，并且新的竞争者以25嘀嗒节拍的间隔添加，市场的速度是25/10=2.5，因为每一个竞争者增加2.5条链路。在变量nNewLinkInterval和nNewNodeInterval中输入程序NetGain.jar的参数，市场的速度是：

市场速度 = nNewNodeInterval/nNewLinkInterval

图12-6总结了通过程序NetGain.jar仿真取得的结果。对于 $m = 100$ 条链路，100个消费者以及3、5和10个初始竞争者，对市场份额的固定点值与 $\log_2$（市场速度）作图。涌现的过程以一种速率定期将链路连接到随机选择的竞争者，以另外一个速率定期创建一个新的竞争者。市场份额是对这些速率之比取对数画出的。

图12-6中的图形被分为两部分：左半部分（负 x 轴）代表一个缓慢的市场，右半部分（正 x 轴）代表了快速市场。一个快速市场要比竞争者更快地增加链路：

快速市场：$\log_2$（市场速度）>0

在一个新兴的市场中生存的竞争者数随着 n_p 增长迅速降低。二项式分布的影响是明显的，因为一开始的大范围的竞争者倾向于在更多的竞争者上扩散链路。消费者被链接到所有竞争者

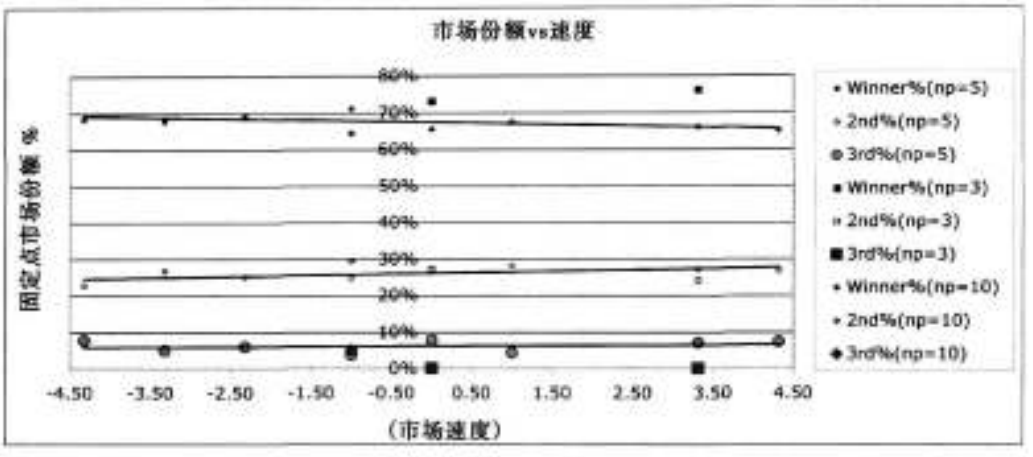


图 12-6 新兴市场中前三个竞争者的固定点市场份额。市场份额与市场速度的对数的关系图证实了对于 $3 \leqslant n_p \leqslant 10$ 时二项式分布固定点市场份额值

节点的概率会更低。这降低了领先者的固定点市场份额。相对于小范围区域的竞争者，大区域中竞争者更难形成垄断。

此外请注意，在一个缓慢的市场中第一个倡议者很容易被一个快速追随者所替代，因为缓慢的市场会给模仿者大量的时间来赶上创新者。在新兴市场初期，一条或两条链路的变化就可以更改以节点度为基础的领导地位。重联的链路是随机选定的，因此有可能使模仿者的链路数量“意外地”超越创新者的。在一个缓慢的市场中，在引入的新链路之间会有许多重联步骤。这意味着在早期涌现阶段领导的地位可以有许多变化。

然而有趣的是，新兴的市场网络趋于变成三路寡头垄断，其中领导者仅因为最早推向市场而近似取得垄断。该结论对于两人兰切斯特策略的实习者来讲一点都不奇怪。兰切斯特策略用于二战以及冷战时期的兵力分配上[⊖]。日本商业战略家后来在 20 世纪 80 年代加以采用，从而导致形成新的兰切斯特策略（Yano，1990）。

根据新的兰切斯特策略，当竞争者有 41.7% 的市场份额时，它在一个倾覆点达到平衡。一旦竞争者达到 41.7%，将来的份额可以向两个方向发展——要么兴起到一个垄断者要么衰落成一个落伍者。在两方竞争中，领先竞争者可以取得 73.88% 的垄断份额，或下降到落伍者状态 26.12%。新兰切斯特策略警告不要尝试获得超过 73% 的市场份额，因为垄断会激发更多的竞争并降低该组织的有效性。它也提醒要防止快速下降到低于 26% 的份额，因为这意味着你即将成为被淘汰的受害者。

这些新兰切斯特策略解释了两个竞争者的市场动态特性，但它没有归纳推广到 $n_p \geqslant 3$。这里提出的模型很好地匹配新兰切斯特策略，并建议三个主要竞争者总是从 $n_p \geqslant 3$ 竞争者的领域中涌现。图 12-6 的仿真结果符合新兰切斯特策略，并表明不超过两位领导者可以生存于同一个新兴的市场中。

一般来讲，新生的 NetGain 网络收敛到一个稳定的固定点市场份额如下：

1. 当 $n_p = 2$，领导者将以约 73% 的市场份额涌现，其他竞争者将仅以 27% 涌现。

2. 不管创新者和模仿者的数量，$n_p \geqslant 3$，经过充分的时间，只有 3 个主要竞争者出现，前两名分别会达到至少 63% 和 26% 的市场。此外，增加 n_p 减少领导者市场占有率，并增加了更多的

⊖ 兰切斯特（Frederick William Lanchester，1868—1946）于 1916 创建了第一个战争数学模型，如《战争中的飞机，第四种武器的到来》一书中所述。

竞争者的生存概率。

3. 随着市场速度的下降，倡导者的优势会减少——缓慢的市场就可能被快速追随者所垄断。市场越慢（添加新的链路速率较低），模仿者成为市场领导者的机会越多。缓慢市场有利于现任者。

12.5 创造性破坏网络

至此，我们假设所有的产品、服务和观念具有相同的感知价值。事实上，消费者是根据对比产品、服务或观念的许多属性进行购买决策的，我们将许多性能笼统指定为*产品价值*。产品价值可与价格、品质、时尚或心理的价值联系到一起，这可能与其内在价值没有多大关系。不管消费者如何排名竞争的产品，他们决定购买一个产品，不可避免地归结为一种基于产品感知价值的*价值主张*。

竞争者获得或失去市场份额与随着时间而变的产品价值有关。某公司可能通过降低它的电视机价格来获取市场份额，增加其性能或增加其广告，使电视机更具有吸引力。在资本主义制度下，竞争者通过不断提高他们的价值主张来竞争。没能跟上不断提高的步伐就会导致不断地失去市场份额，最终导致竞争者的消亡。

显然，当一个竞争者成为市场领导者时，他对产品价值主张改善的能力可能下降。人们可能会争辩说，成功会导致市场领导者规模的增加和相应敏捷性的减少。随着组织成长，他就需要更多的时间来作出决定。当一个组织变得越来越成功，它会变得厌恶风险，并选用了较为保守的改进策略。这个负担可能会导致领导者的产品价值主张的减少。

我们认为，一个市场领导者将增加其利润边际就是因为它能够——毕竟它是市场领导者。作为现任者，无论价格上升或质量、性能下降，领导者利用其品牌名称来锁定客户。即使竞争者正在失去市场份额，利润也可能会增加，因为销售的减少会使利润上升。这种策略不可能会走得太远，导致没有回报和市场领导者的灭亡。

不论出于我们可想象出的何种原因，这都是一个观察得到的事实，当市场份额增长时，竞争者的创新速度趋于放缓。例如，克里斯滕森（Clayton Christensen）将这一放缓归结于*创新者的困境*，他指出大的已经建立的市场领导者沿着持久不断改进的技术路径，这使得他们很容易受到一种新兴科技以外的创新的打击（Christensen，2003）。

与此相反，为了维持一个安全和有保障的技术路径，创新者沿着*颠覆性的技术路径*，其特点是低性能、低价格，重点集中于市场的不足部分。创新者行动迅速以及迅速改善其技术性能的能力，意味着其创新的产品很快超过了现任者提供的价值主张。市场领导者在了解它之前，其市场已经消失——创新者会重新启动该过程。

什么样的价值主张会驱动消费者转到一个新的产品、服务或观念，而抛弃老的呢？我们了解到，某种忠诚是以“粘滞性”的形式存在于随机和新兴市场中。粘滞性往往是由于*高转换成本*，而使消费者不愿意改变。例如，由于陡峭的学习曲线，消费者可能不希望使用新的计算机或软件应用程序，或因为喜好旧车的风格可能会抵制购买一辆新车。消费者继续使用现有的产品可能更容易一些，仅仅是为了避免支付有形或无形的转换成本。因此，现任者可能会守住了部分市场甚至在被一种颠覆性的技术破坏时也是如此。

但是历史已经证明，转换成本最终会被克服解决，整个行业会向颠覆性技术屈服。从化学摄影到数码摄影，从主机到个人计算机，从专有网络协议到 TCP/IP 协议，从唱片到数字 MP－3 音乐播放器，这些都是著名的颠覆整个行业的例子。在每一种情况下，挑战者以较低质量、较低性能的小生境（niche）产品开始，迅速改进，直到它超出了消费者的最低价值主张为止。

不管出于什么原因，市场领导者和垄断是注定要衰落的，会被新的创新所替换。市场领导者的兴衰，是资本主义的自然结果而不是例外。Schumpeter 称之为创造性破坏。它解释了如何通过竞争和一连串创新来使资本主义延续下去。我们可以理解和欣赏 NetGain 网络中创造性破坏如何工作，这需要考虑另一种简单的涌现模型——创造性破坏模式。我们证明了网络中创造性破坏像从前一样导致一个稳定的固定点，但领导者的固定点市场份额是市场份额最低阈值的二次函数，它标志着价值主张的增加。对于这里描述的平方根律，最佳的转换阈值约为 27%/2 = 13.5%。

12.5.1 创造性破坏的涌现

Schumpeter 可能是第一个观察到资本主义制度中创新 - 领导者 - 衰落周期的人。他的创造性破坏理论描述了这样一个循环过程：一个早期阶段的创新，随后它上升到顶部，再后它随着市场领导者遇到了一个新的创新挑战者而最终衰落。在 Schumpeter 的世界观中，创造性破坏“不断地从内部革新，不断地摧毁旧的，不断地创造新的”（Schumpeter，1942）。创造性破坏是资本主义制度生活的本质。

在下面的段落中，我们基于客户的价值主张设计开发出创造性破坏的 NetGain 网络模型，将高价值的产品以低价值产品的价值费用吸引消费者。转换仍旧受到偏好连接的影响，但此外，消费者也会衡量竞争者的产品的价值主张。如果竞争者的价值主张与市场份额相结合的地位低于一个挑战者的，消费者会转换到一个更好的选择。

竞争者认识到这一点，所以他们向上调整自己的价值主张以便于竞争。这可以用许多方式来完成，例如降低产品售价，提高质量等。但竞争者也很贪婪——他们攫取最大限度的利润，这迫使他们在可能的情况下向下调整自己的价值主张。只要垄断能够在市场份额中保持领先，为什么还要牺牲利润呢？领导者在盈利的同时试图减少其产品价值并冒着损失份额的风险。但是如果市场份额开始下滑，领导者必须采取提高其产品质量、价格或广告预算等方式增加价值，这都是以牺牲利润为代价才能实现的。

另一方面创新者以有吸引力的价值主张攻击领导者，旨在破坏市场领导者。没有什么可损失的，反而可能会得到一些市场份额，创新的挑战者在短期内为了市场份额而牺牲利润。它通过给消费者一个更好的交易——更高的质量、更低的价格、更多的广告来实现。在极端情况下，挑战者通过亏损销售来“买进市场份额”——一种可以扭转销售领导地位的短期策略。

在这种 NetGain 竞争博弈中，让挑战者和领导者节点的市场份额作为他们成功的度量，每个竞争者节点的价值——用一个正数——代表感觉到的竞争者产品的价值主张。一方面，每个竞争者想要增加其市场份额，而另一方面，他希望增加利润。一般来说，牺牲价值主张可以增加利润，牺牲利润可以增加价值。因此，两个目标之间存在一个拔河现象——获得市场份额还是最大化利润。

假设竞争者的节点度（市场份额）的变化正比于现任者价值与挑战者价值之比乘以现任者的度：

$$\Delta \text{degree}(\text{challenger}) = \frac{V_{\text{incumbent}}\,\text{degree}(\text{incumbent})}{V_{\text{challenger}}}$$

类似地，当挑战者的度增加时，现任者的度会减小，与价值的比例成正比：

$$\Delta \text{degree}(\text{incumbent}) = \frac{-V_{\text{challenger}}\,\text{degree}(\text{challenger})}{V_{\text{incumbent}}}$$

将第一个方程除以第二个方程产生比率：

$$\frac{\Delta \text{degree}(\text{challenger})}{\Delta \text{degree}(\text{incumbent})} = \frac{-k\,\text{degree}(\text{incumbent})}{\text{degree}(\text{challenger})}$$

其中 $k = [V_{\text{incumbent}}/V_{\text{challenger}}]^2$。换句话来讲，度的改变与价值主张差的平方成正比。或者说，

价值主张的变化与度的平方根成正比。这是创造性破坏的平方根律。

我们只对使一条链路从现任者转换到挑战者所需的转换阈值感兴趣。当这种转换发生时，挑战者的度会有一条链路增加而现任者的度会有一条链路减少：

$$\Delta\text{degree(challenger)} = +1$$
$$\Delta\text{degree(incumbent)} = -1$$

带入比例方程中得出：

$$-1 = \frac{-k\ \text{degree(incumbent)}}{\text{degree(challenger)}}$$

替换 k，求解 $\eta = V_{\text{challenger}}/V_{\text{incumbent}}$ 得出转换阈值：

$$\eta = \frac{V_{\text{challenger}}}{V_{\text{incumbent}}} = \sqrt{\frac{\text{degree(incumbent)}}{\text{degree(challenger)}}}$$

当保持如下价值主张时，消费者从现任者转换到挑战者：如果 $V_{\text{challenger}} > \eta^{*} V_{\text{incumbent}}$，那么

$$\eta = \sqrt{\frac{\text{degree(incumbent)}}{\text{degree(challenger)}}}$$

例如，假设挑战者和现任者具有相同的市场份额：degree（challenger）= degree（incumbent）。消费者将根据产品的价值来决定转换，$V_{\text{challenger}} > V_{\text{incumbent}}$。否则，就不发生转换。这是有意义的，因为所有其他因素都被认为是平等的，所以消费者的决定是基于产品的价值。

现在假设相反：产品的价值是相同的，$V_{\text{challenger}} = V_{\text{incumbent}}$，所以只有当挑战者比现任者有更多链路即当 degree（challenger）> degree（incumbent）时转换才会发生。在这种情况下，模型将恢复到较早研究的随机和新兴市场的模型下的偏好连接的微规则。

创造性破坏是一个涌现过程，一个新兴的市场从一个小生境成长为前面研究过的具有寡头垄断的主流市场。领导者具有垄断地位，例如67%～72%的市场份额，领先的模仿者具有一个参与者的地位，例如26%～28%市场份额。一旦达到这个固定点，将取代现任者的唯一战略是价值主张的变化。这可能是由颠覆性技术规则或某些其他手段来完成的。如果没有破坏，挑战者必须通过降低利润、采用较高的研究和开发成本或更高的广告成本等方式来购买市场份额。

现任者可放弃一些市场份额以得到利润最大化。例如，有67%～72%的市场份额的垄断竞争者能承担得起降低其产品的价值主张，以增加其利润。在其位置上感觉安全，在其产品价值上收费而获利，因此现任者产品的价值开始下降。在某一点，获利可能会导致地位受侵蚀，利用它可以战胜现任者。挑战者就有可能成为市场领导者，或市场领导者通过牺牲利润购买失去的市场份额可能会恢复其领导者角色。

如果一个市场领导者面临一个颠覆性技术，那么它可能因为相关的一些困难而永远无法恢复其领导角色。在这些挑战中普遍缺乏技术性（专利，过程，员工能力）。在某些情况下，例如由大型机迅速过渡到个人计算机，现任者的商业模式过时到一定程度——不管现任者扭转必然结果的努力有多大都会屈从于 Schumpeter 的创造性破坏。

创造性破坏的模型是一个完全基于市场份额的先期模型的超集。它通过添加以下步骤实现了偏好连接和平方根交换律：

创造性破坏涌现

1. 最初给竞争者节点分配相同的值，例如1.0。

2. 让 nPlayer 等于需要开始创造性破坏的最小市场份额，并让 nMonopoly 等于市场领导者因为创造性破坏而开始下降之前最大允许的市场份额。

3. 无限地重复，或直至用户停止为止：

   a. 求竞争者的最大市场份额值，max_share。

b. 如果 max_share > nPlayer，调用创造性的破坏，否则跳过。

i. 创造性破坏：对于所有的竞争者：

（1）如果市场份额 < nPlayer，增 1。

（2）如果市场份额 > nMonopoly，减 1。

这种方法不会进行实际链路的重联——重联在其他地方实现。相反，它根据市场条件对所有竞争者的价值主张重新估价。比最低的市场份额少的落伍者保持不变，但具有至少最小的参与者阈值的参与者会增值以获得更多条链路，而具有比最大市场份额多的领导者会减少其价值以最大化利润。

创造性破坏之所以发生是因为竞争者争夺数量有限的消费者的结果，这些消费者（由链路表示）被吸引到具有最好价值主张的竞争者上。我们以方法 NW_doCreativeDestruction( ) 来进行仿真，每个时间步调用该方法以便确定现任者是否应降低其价值而挑战者增加其价值。竞争者增加其价值以赢得市场份额，而垄断减少其价值以最大化利润。

输入参数 nPlay 和 nMonopoly 建立何时会增加价值或利润最大化的阈值：nPlayer 代表了较低端的市场份额，通常为 26%，nMonopoly 代表了更高端的市场份额，通常为 72%。当其市场份额超过 nMonopoly 时，现任者降低其产品的价值，而其市场份额低于 nPlayer 时，挑战者增加其产品价值。竞争者节点的市场份额等于节点的度除以 NetGain 网络中链路的总数。一个成熟的市场被定义为一个至少有一个竞争者节点的市场份额超过 nPlayer 的 NetGain 网络。因此，该方法首先必须找出高价值的竞争者节点，以确定是否需要调整值。如果不需要调整，该方法什么都不做。

```
public void NW_doCreativeDestruction(){
    double max_marketshare = 0.0;
    double market_share;
    for(int i = nConsumers; i < nNodes; i++){      //Find largest %
      market_share = (float)nodes[ i].degree/nLinks;
      if(market_share > max_marketshare) max_marketshare = market_share;
    }
    if(max_marketshare >= nPlayer/nLinks)          //Mature market
      for(int i = nConsumers; i < nNodes; i++){
        market_share = (float)nodes[ i].degree/nLinks;
        if(market_share <= nPlayer/nLinks && market_share > 0){
           nodes[i].value++;                       //Buy market share
        }
        if(market_share >= nMonopoly/nLinks && nodes[ i].value >= 1){
           nodes[i].value- -;                      //Increase profits
        }
    }
}//NW_doCreativeDestruction
```

同时，竞争者重新估计其产品价值，以吸引消费者或最大化利润，链路根据每一个竞争者所提供的价值主张和前面描述的转换阈值进行重联。该过程由 Java 方法 NW_doCompete( ) 仿真，它有两种模式：新生和非新生。这种方法是随机涌现过程的扩展，并且实现了两个额外的功能：（1）它在 nNewNodeInterval 时间间隔添增新的竞争者，直至所有 n_p = nInputCompetitors 已被添加为止；（2）它在 nNewLinkInterval 时间间隔添加新的链路，直到所有的 m = nLinks 链路已被添加为止。因此，市场的速度为 nNewNodeInterval/nNewLinkInterval。例如，如果一条新的链路以 10，20，30，…时钟嘀嗒添加，而一个新的竞争者节点以 25，50，75，…时钟嘀嗒添加，则市场的速度则为 25/10 = 2.5。这个相对快速的市场对每一个新的竞争者增加 2.5 条链路。

这种方法在价值主张的基础上实现了从创造性破坏中导出的转换阈值：即名义上当 challenger. value > （threshold） incumbent. value 时进行重联链路。阈值是现任者与挑战者节点度之比的平方根。转换发生既可以是因为挑战者具有一个很大的市场份额，也可以因为它的价值主张比现任者的价值主张更大。

```
public void NW_doCompete(int t, boolean doNascent) {
  if(nInputConsumers <= 0 || nInputCompetitors <= 0) return;
  int consumer_node = (int)(nConsumers*Math.random());
  int challenger_node = nConsumers+(int)(nCompetitors*Math.random());
  //Add competitor with one link
  if(doNascent && nCompetitors < nInputCompetitors
       &&
    t == nNewNodeInterval*(t/nNewNodeInterval)){
       NW_doAddNode(Long.toString(nNodes));
       nCompetitors++;
       NW_doAddLink(nodes[consumer_node].tag, nodes[nNodes-1].tag);
       Message = "New Competitor:";
    }
    //Add link
    if(nLinks < nInputLinks && challenger_node >= nConsumers
       &&
     t == nNewLinkInterval*(t/nNewLinkInterval)){
     if(NW_doAddLink(consumer_node, challenger_node))
        Message = "New Customer:";
    }
    if(nLinks <= 0) return;
    //Switching threshold
    int any_link = (int)(nLinks*Math.random());
    Link e = Links[any_link];
    Node consumer = nodes[e.from];     //consumer node -> competitor node
    Node incumbent = nodes[e.to];      //links always point to competitor
    //pick a challenger at random
    challenger_node = nConsumers+(int)(nCompetitors*Math.random());
    Node challenger = nodes[challenger_node];
    double denominator = challenger.degree;
    if(incumbent.degree <= 0           //Eliminate competitors with no share
            ||
       denominator <= 0) return;       //Eliminate divide-by-zero error
   double threshold = Math.sqrt((incumbent.degree)/denominator);
   if(challenger.value > threshold*incumbent.value
         &&
      challenger.degree < nMaximumDegree) {            //Switch
      if(NW_doAddLink(consumer.tag, challenger.tag)) { //Avoid duplicate
         NW_doCutLink(any_link);
         Message = "Switching to challenger:"+
              "[degree(challenger) ="+Long.toString(challenger.degree)+
              ">"+Long.toString(incumbent.degree)+" = degree(incumbent)]";
   }
  }
}//NW_doCompete
```

12.5.2 平方根律固定点

当竞争者为了吸引更多的市场份额更改其价值主张时会发生什么？直观地讲，市场份额应该在竞争者之间摆动，首先挑战者增加其价值主张，直到达到了领导地位为止，而领导者则获取利润。当情势逆转时，现任领导者也必须做同样的调整。因此，我们期望领导者地位的变换不断地重复下去。但是实际情况却并非如此。

如果我们从一个随机网络开始，并使用了上述的创造性破坏平方根算法，稳定的非随机网络将从随机网络涌现。涌现通过两个阶段控制随机网络。首先，随机网络将像前面一样确定领导者、参与者和落伍者。通常情况下，按照上一节导出的淘汰模型，$n_p \geq 3$ 个竞争者的域将会缩小到3或4个节点。然后，其余的竞争者将在几个周期内交换领导者位置，最终达到一个稳定的状态。

前两个竞争者的价值振荡如图 12-7 所示。注意，每个波的振幅稳定下降，直至达到零为止。当这种现象发生时，该网络已达到它的固定点。收敛可以很慢，程序 NetGain. jar 可能需要多达20万个时间步才能达到一个稳定的固定点，尤其是如果最低阈值 nPlay 和垄断阈值 nMonopoly 总和为 100% 时更是如此。

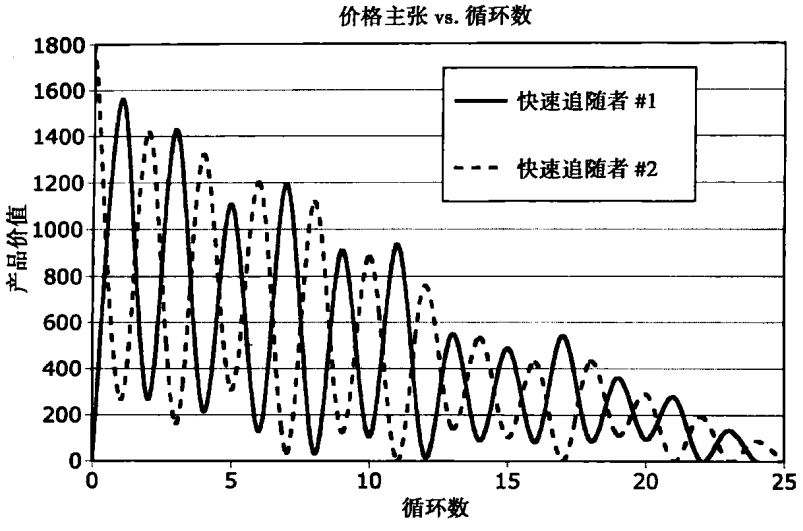


图 12-7　在创造性破坏的第二阶段——淘汰前两个竞争者的值。最初，随机网络对于所有竞争者都具有 100 个消费者、10 个竞争者和 100 条链路，并且价值主张为 1.0

在随机网络中使用平方根算法的竞争者之间的竞争中，两个或三个竞争者迅速成为主导，这是因为偏好连接的增加回报的属性。一旦竞争者超越最低阈值，成为一名参与者，也就是说 27%，它就会开始增加自己的价值主张获得更多的市场份额。当快速追随者达到垄断阈值 nMonopoly，它做出相反的调整，降低其产品的价值以便增加利润。当发生这种情况时，新的领导者就会面临第二个快速追随者的挑战，并且过程重复。不过，这并不是无限地重复！

在创造性破坏涌现的第二阶段，前几个竞争者之一慢慢开始赢得其他对手。每个价值主张周期振幅略低于前一个，但是需要更长的时间才能达到。但是迟早，竞争者的价值将达到零。所有价值主张大于零的其他竞争者的节点将继续竞争，但最终，另外一节点也会达到零值。当所有幸存的竞争者节点达到零，就达成了固定点。

当固定点涌现时，生存下来的竞争者的市场份额价值是多少？这个问题的分析解是目前一个未解决的研究问题，所以我们要靠经验来解决这个问题。考虑一个初始化为 100 条链路、100 个消费者、10 个竞争者的随机网络。运行程序 NetGain. jar 较长一段时间以便产生固定点市场份额的“典型运行”结果，也就是 72% 和 60% 的垄断阈值。让最低参与者的阈值从零到 50% 不等，并为每个幸存的竞争者绘制平均网络的稳定固定点市场份额，如图 12-8 所示。这些平均数随最小参与者阈值而变化。然后建立一个如图 12-8 所示的实线近似模型，并比较这两者。

图 12-8 中所示的“鱼状”曲线是通过将固定点市场份额近似成最低参与者阈值函数获得的。当最小参与者阈值加上垄断阈值超过 99% 时，除了一个竞争者之外所有固定点市场份额都为零。领先的竞争者最终以 100% 的市场份额结束。从图 12-8 中看这似乎像“鱼尾巴”。一个高的最小参与者阈值并没有多大好处，因为它总会导致垄断。一般来说，由创造性破坏算法来设定使用的垄断阈值上限，即竞争中幸存竞争者的数量：

$$\text{最小参与者阈值} < (1 - \text{垄断阈值})$$

现在考虑如下最小参与者阈值的范围（1 - 垄断阈值）。经验结果显示，前两个竞争者的固定点市场份额遵循二次关系形成“鱼的躯干”。的确，我们通过注意到前两个竞争者大致服从前面研究过的 Bass 和 Fisher-Pry 一阶导数方程来近似这种关系。回想一下，Bass 和 Fisher-Pry 状态方程的右边是一个市场份额的二次函数。通过借用 Bass 和 Fisher-Pry，我们断定领先竞争者的固定点也服从该规则。

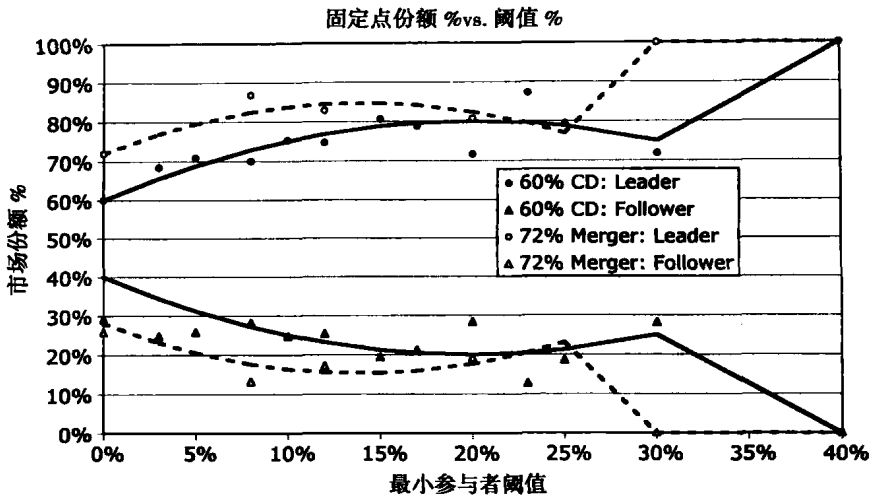


图 12-8 两个涌现创造性破坏网络的市场份额固定点与最小参与者阈值关系——其中一个仅有创造性破坏的网络具有60%的垄断阈值，另一个网络具有创造性破坏加上企业并购和72%的垄断阈值。所有网络链路含有100条链路和100个消费者节点，并从 $n_p = 5$ 个竞争者开始

我们将领导者的固定点市场份额近似为一个最低参与者的阈值函数，如下：

$$\text{Market\_share\_leader}(m,p) = m + rp(a - p)$$

$$\text{Market\_share\_follower}(m,p) \leqslant 1 - \text{market\_share\_leader}(m,p)$$

其中

m = 垄断阈值；nMonopoly

$0 \leqslant p \leqslant (1-m)$ = 最小参与者域值，nPlayer

r = 链路重联速率参数

$$a = 1 - m$$

代入通过曲线拟合得到的常数，并注意 $a = (1-m)$，我们获得以参与者和垄断阈值表示的近似值（消除一个参数 a，并在这种情况下设置 $r = 5$）：

领导者的固定点市场份额

$$\text{Market\_share\_leader}(m,p) = m + 5p(1 - p - m)$$

其中 m = 垄断阈值，p = 最小参与者阈值。

例如，让 $m = 72\%$，$p = 10\%$，我们得到：

$$\text{Market\_share\_leader}(0.72, 0.1) = 0.72 + 5 * 0.1 * 0.18 = 81\%$$

$$\text{Market\_share\_follower} = 100\% - 81\% = 19\%$$

对垄断和参与者分别运行5次仿真得到的平均固定点为83%与17%[⊖]。考虑到仿真数据方差较高，对于每个数据点只获得了5个样本，这种情况下近似方程是相当好的。

使用平方根算法的创造性破坏，如果最小参与者阈值保持很小就会导致双寡头[⊜]。对于这里进行的仿真，参与者阈值必须小于27%（假设72%的垄断阈值）。更有趣的结论是，即使我们从5个竞争者开始也会涌现一个双头垄断。这一结论成立，与竞争者最初的数目（$n_p > 3$）无关。最小参与者阈值降低会提高弱者的生存概率，因为这意味着早在这个过程开始，失败者就增加

⊖ 图12-8的原始数据是对每个数据点运行5次的平均。

⊜ 垄断可能以非零概率涌现，三个幸存者也是如此，但是这些事件的概率很低。

其价值主张。阈值越高，失败者开始调整其价值适应实际市场就越晚。

最低阈值和固定点间的关系不是线性的。图 12-8 中显示了追随者的 U 形市场份额曲线与市场领导者的倒 U 型曲线。对于图 12-8 中所使用的参数，最佳的最小参与者阈值位于零至由程序 NetGain. jar 的用户设定的最小参与者阈值中间。当 m =72% 时，最佳参与者阈值是什么？这留给读者作为练习。

通过这些有限的数据来概括太多结论可能是危险的。图 12-8 中显示的实验结果取决于 NetGain 网络的初始参数。我们假设一个随机初始网络有 100 个消费者节点、100 条链路和 n_μ = 5。起初，竞争者开始值为 1.0。读者尝试其他初始值的组合。这些结果为进一步研究创造性破坏的涌现行为提供了很好的铺垫。

12.6 企业并购网络

按照 Schumpeter 的说法，创造性破坏是竞争的必然结果。人类的独创性和网络效应使其不仅可能，而且也是不可避免的。另一方面，企业并购在竞争者中提供了更直接的控制市场份额的方法。当两个竞争者合并成一个时，希望合并后的实体提高其价值主张，使新业务更具竞争力。而这在所有的情形中可能是正确的，也可能是不正确的，兼并的确会增加市场份额，因为至少在一段时间内，合并后的实体的市场份额等于竞争者的市场份额的总和。因此购入竞争者是增加竞争者在市场上的地位的途径之一。那么是否可以采用与其他落伍者兼并的策略而使一个落伍者成为一个领导者呢？在本节中我们将测试这种假设，学习兼并是很有帮助的，但是这不能保证可以上升到资本主义食物链的顶部。

考虑以下 NetGain 算法的微规则，当市场份额的百分比总和低于最小参与者阈值时，将两个节点合并在一起。这个想法很简单，如果两个落伍者合并，他们可以通过合并其消费者而从落伍者跨越成为参与者。这种飞跃是否会推动合并后的竞争者达到参与者地位呢？希望新合并的竞争者能够通过增加回报谈判其突然增加的市场份额而变成更高的百分比。如果这样的话，那么企业并购可成为弱者的有效战略。

企业并购微规则

1. 随机选择两个竞争者 X 和 Y，设置 X. share 和 Y. share 分别为竞争者 X 和 Y 的市场份额。
2. 如果 X. share + Y. share < 最小参与者阈值，那么
   a. 将更少市场份额的节点比如 X 合并加入到有更多市场份额的节点比如 Y 中：
      i. 将 X 的所有链路复制给 Y；
      ii. 删除节点 X。

请注意，合并后的竞争者的价值主张仍然是相同的——只有度，以及因此合并后的节点的市场份额变化。新创建的节点应该以更大的速度增加其市场份额，如果其新获取的市场份额驱使它超出了垄断阈值，它也就变成了新的现任领导者。这种说法在下一节中详细研究。

12.6.1 合并节点的 Java 方法

下面的 Java 方法实现企业并购微规则。方法 NW_doMerge( )解释上面指定的微规则。它保证两个截然不同的竞争者节点是随机选择的，并决定是否两个市场份额之和足够小以保证两者可被合并。方法 NW_doMergeNodes(int left,int right)进行从一个节点链路到另一个节点的重联链路的繁琐工作，然后删除较小的节点。它假定右边是较小的市场份额节点，因此所有的链路移动到左边的节点。每一条链路都需要进行检查，如果它指向右节点，那么它就会被重联指向左节点。每一个节点对应的度都进行调整。然而，合并后的节点的值保留不变。

```
public void NW_doMerge() {
        if(nCompetitors <= 1) return;
        int j = nConsumers+(int)(nCompetitors*Math.random());    //Pick one
        int k = nConsumers+(int)(nCompetitors*Math.random());    //Pick another
        while(j == k) k = nConsumers+(int)(nCompetitors*Math.random());
        double ms_left = (float)nodes[ j] .degree/nLinks;
        double ms_right = (float)nodes[k] .degree/nLinks;
        double threshold = (float)nPlayer/100.0;                 //Laggard share
        //Does Merger make sense?
        if(ms_left+ms_right < threshold){
            if(ms_left > ms_right)
                NW_doMergeNodes(j, k);
            else
                NW_doMergeNodes(k, j);
        }
    }//NW_doMerge

 public void NW_doMergeNodes(int left, int right){
     for(int j = 0; j < nLinks; j++){
        if(Links[j] .to == right) {
            Links[j].to = left;
            nodes[left].degree++;
            nodes[right].degree- -;
        }
     }
     NW_doEraseNode(right);
 }//NW_doMergeNodes
```

12.6.2 合并加速创造性破坏

图12-8显示了将创造性破坏和合并算法相结合的结果。虚线和空数据点对应于垄断阈值72%的创造性破坏及企业并购所有都同时运行的情况。在每种情况下，为了使竞争者幸存下来，涌现网络达到一个固定点市场份额值。这些固定点是运行5次模拟的平均，并沿着纯创造性破坏的数据作图，这样两者便于进行比较。

我们像前面一样得到了一张“鱼形图”，但有两个主要不同：（1）创造性破坏和兼并过程结合起来的鱼形图被移到了纯创造性破坏的左边；（2）结合起来的过程的振幅稍高。第一个不同主要是由于合并，而第二主要是由于垄断阈值较高（72%比60%）。

合并具有明显提高速度的效果，有了它不可避免淘汰的发生。这意味着，随着最小参与者阈值的增加，图12-8中的虚线上升和下降得更快。仿真早晚也会收敛于网络的固定点——5到10倍之快。这意味着合并过程加快了较弱竞争者的淘汰，最小参与者阈值越低，淘汰的速度就越快。实际上，合并加速了这种必然。

第二个区别也许是有合并与无合并创造性破坏之间的第一个区别的后果。但它也是更高垄断阈值的后果。市场领导者固定点份额模型稍微加重了一些，因为$r=6.5$，而不是5：

$$\text{Market\_share\_leader}(m,p)=m+6.5p(1-p-m)$$

例如，让$m=72\%$，$p=12\%$：

$$\text{Market\_share\_leader}(0.72,0.12)=0.72+6.5*0.12*0.16=84\%$$

$$\text{Market\_share\_follower}=100\%-84\%=16\%$$

经过5次模拟运行垄断和参与者的平均固定点分别是83%和17%。再次，考虑到模拟数据的方差较高，并且对每个数据点只获得5个样本，这种情况下近似方程是比较好的。

企业并购通过尽早而非晚一点消除较弱的竞争者加速了创造性破坏过程。虽然合并常用于提高市场份额和现实世界中的经济规模增益，兼并一般不改变不可避免的固定点。然而，这里完成的模拟支持数值结论，但没有告诉我们最终哪个竞争者会成为领导者。这种分析只是告诉我们将要涌现一个领导者。在商业现实世界中，我们也有兴趣预测哪个竞争者会获胜。企业并购可

以改变一个竞争者的领导地位。而预测哪个落后者因兼并最终将成为优胜者的问题留给读者来解决。

练习

12.1 在图 12-3 中单一产品网络的垄断竞争者需要多长时间才能上升至 50% 的市场份额?

12.2 使用习题 12.1 的结果，将 Bass 方程运用到图 12-3 的网络得到的市场份额评估是什么?

12.3 在一个 $n_c = 100$、$m = 350$ 的多产品网络中，并且参数如表 12.1 所示，那么在 $n_p = 5$ 个竞争者中固定点市场份额的大致分布如何?

12.4 执行类似于已进行的获得图 12-4 和表 12-1 中数据的实验。让 $n_p = 3$，m 的范围从 100 到 200 条链路。问：

(1) 市场份额遵循幂律分布吗?

(2) 涌现达到它的固定点以后还有多少竞争者留下来?

(3) 当 $m = 200$ 时，对于三个竞争者的固定点市场份额是多少?

12.5 在一个随机 NetGain 网络中，$m = 100$ 条链路，$n_p = 4$ 个竞争者，$n_c = 50$ 个消费者，什么情况下消费者节点有四条链路? 什么情况下它至少有 3 条链路?

12.6 当新链路之间的间隔是 100 并且新的竞争者之间的间隔是 20 时，一个新生 NetGain 网络市场的速度是多少?

12.7 新生 NetGain 网络总是会产生两个或三个寡头竞争者吗? 如果是这样，为什么?

12.8 在一个创造性破坏涌现中的领导者以最多市场份额结束的情况下，最小参与者阈值的最优值是多少? 假设垄断阈值，$m = 60\%$，$r = 5$。

12.9 在创造性破坏的涌现中，当平方根律有利于挑战者时，消费者转向挑战者。假设现任者的产品价值是 100 元，如果现任者有 72% 的市场份额而挑战者有 27%，那么挑战者必须采用什么价值才能使消费者转向呢?

12.10 当 $n_c = 50$，$m = 100$，$n_p = 10$，垄断阈值 = 72%，最小参与者阈值 = 10% 时，有多少竞争者通过合并幸存于漫长的创造性破坏涌现呢?

第13章
Network Science: Theory and Applications

生 物 学

网络科学近来已应用于生物系统各个层次的研究中（Albert, 2007; Jeong, 2000）。例如，食物链就是相互依存的生态系统的高级网络模型。节点就是物种，而链路就相当于竞争生物中的猎物－捕食者关系。在生物学中的另外一个极端，细胞中的蛋白质、DNA、mRNA（信使 RNA）的生物化学过程也会形成网络。这些网络类似于微电子芯片上的电子电路，但这里却不是用来调节电子的流向，蛋白质表达网络是描述蛋白质、氨基酸和其他代谢产物与蛋白质之间交互作用的规律。在这种情况下，节点为 DNA、mRNA 和蛋白质，而链路表示调节化学的质量流量。

该领域很新颖并仍处于发展之中。迄今为止，网络科学在生物领域中的涌现沿着两条基本途径进行：*静态分析*，其中生物网络的拓扑结构与功能有关，也就是功能服从形式；另一种是*动态分析*，其中一些生态系统上的生化过程等同于流过网络的流或信号。在动态分析中，生物网络的作用是通过状态方程或状态矩阵来建模，这类似于在第 9 章中介绍过的内容。

*静态分析*试图将*度序列分布*、*平均路径长度*、*紧度*（*介数*）、*聚类系数*与有机体的行为联系起来。例如，据称生物网络趋向于构成无标度网络，因为无标度网络在遭受随机攻击或意外事故时不易产生故障。我们在第 11 章中展示了网络是如何被攻击的，以及高度节点确实对于无标度网络的功能来说至关重要。

动态分析会更加困难，在作者撰写本书之时，它仍然是一个处于前沿的研究主题。简单地讲，动态分析试图将微电子中应用的电路设计原理运用于微生物学中。是否有可能发现控制生物网络的规则以解释生物有机体的动态行为？例如，生物网络遵循基尔霍夫定律吗？动态分析或许是网络科学最吸引人的应用。因此，本章在着重解决动态模型时特意留下了很多尚未解答的问题。

在本章中，我们提出如下问题并推测一些有待解决的研究问题：

1. 生物网络是多种多样的，从宏观层次的食物链到微生物层次的单细胞内的蛋白质相互作用网络。进一步来讲，目前还没有有关生物网络的标准定义。从类似电路的布尔网络、贝叶斯网络、影响网络，再到线性、非线性连续系统网络，对此都有各自不同的表述。

2. 生物网络模型大致可以分类为*静态和动态*、*布尔和连续*、*线性和非线性*。在本章中，我们侧重于研究亚细胞级别的微生物连续、线性模型，而且也兼顾连续网络的布尔近似和一些行为上的非线性效应。

3. 很多生物网络显示出带有小世界效应的无标度网络的属性。有人认为功能服从形式——*无标度网络*更能容忍随机性故障，而*小世界效应*能使生物体对小的摄动产生更迅速的响应。我们通过比较随机网络和无标度网络的谱半径，从一种非传统途径来研究这种论断。我们发现链路上的权重——表示转导矢量——对质量动力学网络的谱半径大小至少像其度序列分布那样重要。

4. 已经发现在*蛋白质表达网络*中大量存在被称为*模体*（motif）的某些预联子网：*双扇*（bifan），*双平行*，*前反馈回路*（FFL）和*双前反馈回路*（bi-FFL）。这些模体被认为在生物网络内调整加速或延迟信号，在亚细胞级别上调整启动或停止生化反应链。

5. 我们详细检测两个生物网络模型：线性连续蛋白质表达网络、线性连续质量动力学网络。线性连续网络的状态方程是 $\partial S/\partial t = W^{T}S$，$S(0) = S_0$，质量动力学网络的状态方程是 $\partial S/\partial t = \sum_{j\in \text{inlinks}} \text{inFlow}_j - \sum_{k\in \text{outlinks}} \text{outFlow}_k$，其中 W^{T} 是对应于有向权重网络的转置邻接矩阵，流入/流出矩阵都是从 W^{T} 中获得的。

6. 与连续网络对等的布尔网络近似是一种非线性网络。布尔网络包含取值为布尔值是0或1的节点，并执行布尔操作 AND，OR。链路提供门函数，链路加权 +1 意味着一个信号从一个节点无变化地传送到另一个节点。链路加权 -1 意味着这个信号逻辑上取反：1→0，0→1。

7. 我们通过仿真说明，通过随机重联链路增加熵同时也提高网络 hub 的度，导致生成类似无标度网络的一个涌现过程。这就是为什么会在自然界中找到无标度网络的原因。然而，随机重联链路减小谱半径（因此增加动态稳定性）不足以使随机网络的谱半径减少到所谓的稳定网络的程度。

8. 我们通过仿真证明，质量动力学网络在其输出链路权值总和为1时会更稳定。无标度网络倾向于比随机网络更稳定，但是仿真结果也表明随机网络在其输出链路权值总和为1时更稳定。

本章所述的蛋白质表达网络的动态行为仿真可以由程序 BioNet. jar 来进行，它与 Influence. jar 程序有关。BioNet. jar 支持连续网络模型和布尔网络模型，并且自动计算谱半径、固定点吸引子（近似值），以及随后几节研究的几种涌现过程。

13.1 静态模型

静态分析试图将功能与形式关联起来，或者用网络科学中的术语来讲就是网络拓扑与其功能和行为关联起来。表13.1总结了到目前为止（2008年）所能收集到的证据以支持静态分析（Albert，2007）。虽然这些证据在某些方面上还是推测性的，但它证明了本书先前及其他地方所给出的分析。然而读者需要知道这是一项前沿研究，还有许多未解决的问题。

表13-1 网络属性与生物功能对比

网络性质	生物功能	评 价
无标度	容错	由于随机地移除节点，因此不大可能失败
小世界效应	迅速响应	短路径意味着对环境及冲击变化的响应时间更短
聚类	模块化	有效的层次化设计导致有效的功能性
紧度（介数，中心性）	容错，迅速响应，效率	因为最中心节点也是hub，导致很可能为无标度拓扑结构的结果

13.1.1 无标度属性

Reka Albert 提供了有关无标度网络和蛋白质表达网络功能性关系的调查（Albert，2005）。蛋白质表达网络将蛋白质、DNA、RNA 和其他细胞中的小分子作为节点，信号发送路径和其他分子调节机制作为链路。信号发送和调节的过程是一个转变细胞内化学物质如 DNA、RNA 及其他新陈代谢产物的关键过程。表达级别对应于过程中的某种化学浓度，行为相当于网络的组合和演化。稍后我们将详细研究蛋白质表达网络。

大多数研究是基于非常简单的有机体，如大肠杆菌、酿酒酵母。例如，Albert 报告酵母蛋白质网络是一个带有指数为2.5、聚类系数为 $O(1/\lambda^2)$ 的无标度网络。其他研究人员也报告，在酵母蛋白质网络中具有小平均路径长度的小世界效应和密集的本地邻居。那么这种结构的功能是什么呢？

Giaever 等人观察到有超过73%的酿酒酵母基因是非本质的，并声称删除这些非本质的基因

对细胞的影响较小（Giaever，2002）。这证实了随机基因删除攻击下，酵母基因的健壮性——这里很有可能是移除了不重要的基因。然而，酵母细胞极易受到对高连通 hub 或具有高紧度中心性的节点的破坏。在无标度网络中，hub 最有可能是具有最大紧度的节点。所以这证实了无标度网络健壮性的一般观点。然而目前还不清楚是高连通的 hub 结构还是进化因素，哪个在先，导致了无标度结构。

Zhu、Gerstein 和 Snyder（Zhu，2007）认为生物网络的无标度拓扑主要来自“链路动态特性”，而不是 BA（Barabasi-Albert）生成过程的偏好连接。链路动态特性近似地描述为交互作用损耗和导致了无标度 hub 的偏爱互作用增益。或许链路动态特性只是第 7 章中所描述的 Mihail 生成过程的另一个代名词。回顾 Mihail 等人所提出的涌现过程产生了一个带有度序列分布的预定义拓扑的设计者网络。网络被设计成带有某些度序列分布的涌现，就是因为它们已经在每个节点预联了 stub。

这只不过是猜测而已，但是或许每个蛋白质的生物化学结构决定蛋白质能够支持多少种表达过程。一个高度的蛋白质可能比低度的蛋白质有更多的的受体（键联），因此就能够更高度地连通。如果果真如此，那么在每个节点预联单臂集合的抽象概念就对应于现实世界中的生物化学。

13.1.2 小世界效应

像聚类和短的平均路径长度这种小世界效应被认为是生物网络对摄动迅速响应的一种结果。已经在带有短的和冗余路径的蛋白质网络中观察到新陈代谢和蛋白质相互作用做出的快速而有效的反应，同时也观察到高聚类本地邻居——两种都是小世界网络的属性。它就好像生物网络将自组成尽可能保证短的平均路径长度的响应。

聚类可能是蜂窝网络模块化的结果，即蜂窝网络体系结构遵循从最佳电子电路设计中找到的通用规则。电子电路是以高度模块化结构设计的，以减少总体的复杂性。一个带有很多简单部件的系统要比一个大型复杂的组件要简单得多。同样的思想贯穿于软件的设计，复杂性被模块取代，期望模块是自包含的，具有到达其他模块最少的链接。这经常会导致层次化的设计。

在生物网络中观察到大量的模块，称为模体，这意味着自然界重复不断地应用实践证明是可取的（行之有效的）同一设计规则。一个模体是一个具有良好定义的拓扑的小子图，它被有机体以各种方式使用（参见图 13-1）。将模体想象成为一种复杂的生物系统中的简单组件，它使用电子电路和计算机软件中降低复杂性的同样方式减少网络的整体复杂性。

图 13-1a 的双扇模体表示从随机连通调节网络中找到比预想更丰富的聚类。节点代表蛋白质、DNA 或在细胞内的生化反应涉及的其他代谢产物。带有加号标记的链路代表一种激活反应——实质是前反馈机制，以增加某种代谢化学反应的浓度。一个带有减号标记的链路代表了相反的抑制反应，其本质上是一种负反馈机制。双扇的前反馈信号或表达级别加速了细胞内代谢的浓度。

同样地，图 13-1b 中所示的双平行模体代表连锁反应：蛋白质 A 激活 DNA 的 B 和 C，从而激活蛋白质 D。单一输入和前反馈回路（FFL）模体代表简单的前反馈机制，而双前反馈机制（bi-FFL）更复杂，代表 B、C、D 代谢物的激活由 A 来完成，进一步由代谢物 B 来加速反应。

这些前反馈模体被认为是调节网络内信号的加速或延迟。例如前反馈回路可能是脉冲产生的源头，这些标记代表了细胞内某种代谢的产生。另外，反馈回路被认为是控制状态锁定——实现并保持一定的网络状态。类似于电子电路设计是不可避免的，但是还没有完全理解如何控制这些“化学电路”的动态行为的精确规则。

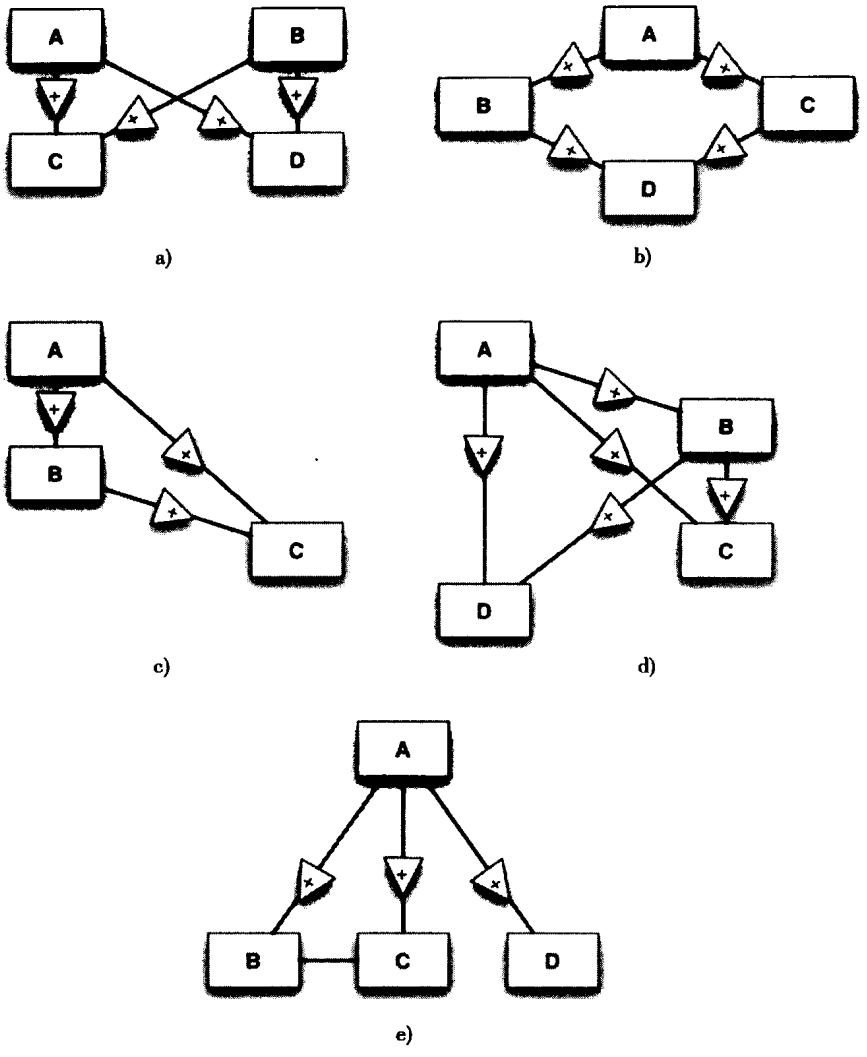


图 13-1 蜂窝网络中的通用模体：a）双扇；b）双平行；c）前反馈回路；d）双前反馈回路；e）单输入

很显然，蜂窝过程是由层次化结构的模块构成的，就像一个精心设计的电子电路或计算机程序那样可分解成子模块等。模块间有高度的串扰和重叠（Albert，2005）。此外，三角形（聚类）在蛋白质网络中非常丰富，带有正的和负的反馈机制。我们从前面的章节里得知，三角形主要负责动态网络中的同步和动态稳定性。那么三角子图在生物网络中是否扮演着类似的作用呢?

13.2 动态分析

支持这种观念的证据越来越多，因为它们的结构而调节网络的行为方式，也就是说功能服从形式。这在微生物系统建模成表达网络中最明显。这种方法的目的是要准确地预测一个生物过程的行为，这和我们预测电子电路的行为一样。下面的分析主要适用于基因表达或蛋白质表达网络，因为这些网络与相关的生物网络比起来已经做了更详细的探讨。不过，这种概念也可以适用于其他生物系统。

基因、蛋白表达网络是一种影响网络，但是有一些显著例外：（1）下一个状态函数有很大的不同，并且在生物网络上经常是非线性的；（2）生物网络充满了抑制反馈回路。这种基本影响或第10章中的I-nets模型的增强大大提高了网络的复杂性。我们首先研究简单的线性连续动态模型，之后再返回到更复杂的非线性动态模型。

13.2.1 线性连续网络

图13-2演示了线性连续表达网络的一些基本构建模块。这种表达网络是第10章中学习过的影响网络的特殊形式。因此，我们将采用与线性生物网络相同的理论。然而，我们的理论不能表达生物网络的非线性行为。回顾一下，当网络的下一状态函数是线性时那么网络就是线性的，当我们使用平滑连续函数来描述状态转换时，那么它就是连续的。

图13-2a的简单的杠铃形蛋白表达对线性蛋白表达网络的最基本的相互作用进行建模。蛋白质A与DNA中的氨基酸键联，导致一些副产品。在表达层次上，mRNA在这种情况下的浓度可能会从负（负反馈）变化到某一正的层次。

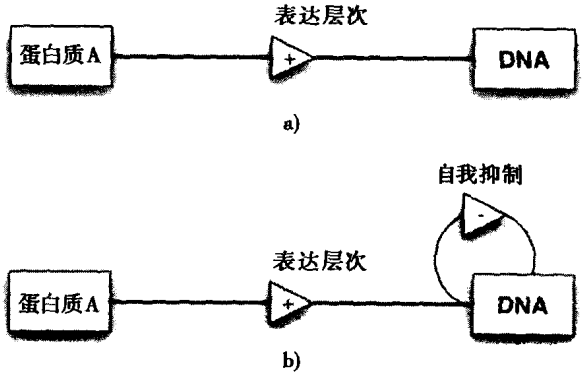


图13-2 基本蛋白质表达网络组件：a）简单表达；b）抑制表达

表达可能由于以下原因出现：转录——将遗传信息从DNA转录到RNA的过程；或翻译——将带有的遗传信息以mRNA（信使RNA）从DNA传递，并将它转变成与肽键键联的一系列氨基酸的过程。这一过程发生在细胞的核糖体中。为了简化起见，我们把转录和翻译同等看待——作为广义生物网络中的加权和有向链路。

简单表达是一个集成函数，如图13-2a所示。若没有某种形式的负反馈，图13-2a中DNA节点浓度级别是无限的。图13-2a中DNA蛋白质A的影响是用差分方程$(\partial / \partial t)\mathrm{DNA} = w(A)$进行数学建模的，其中DNA是DNA副产品的浓度，$w$是一个权重因子，代表表达级别，$A$是蛋白A的浓度。每个节点的最终状态都是由微分方程集成构成的。给定蛋白质A一个常量级别，那么DNA级别将无限地增长。

自动抑制是使用负反馈抑制表达级别的过程。图13-2b中显示的网络是自动抑制的，因为现有的级别控制着浓度级别。在数学建模中，DNA的浓度变化率由两个输入所控制：蛋白质A的加权（常数）级别和DNA的负加权级别：

$$\frac{\partial}{\partial t}\mathrm{DNA} = w_A A - w_{\mathrm{DNA}}\mathrm{DNA}$$

$$\mathrm{DNA}(0) = \mathrm{DNA}_0$$

$$\frac{\partial}{\partial t}A = 0$$

$$A(0) = A_0$$

举例来说，假定蛋白质A的单位级别、单位加权和初始条件如下：

$$\frac{\partial}{\partial t}\mathrm{DNA} = A - \mathrm{DNA}$$

$$\mathrm{DNA}(0) = 0$$

$$\frac{\partial}{\partial t}A = 0$$

$$A(0) = 1$$

此联立方程组有以下解：

$$\mathrm{DNA}(t) = 1 - \mathrm{e}^{-t}$$

$$A(t) = 1$$

随着时间的推移，DNA 的浓度级别从零上升到 1.0，之后就保持不变。超出消失点（称为固定点或奇异吸引子）之外的反馈回路抑制表达，而不是让该节点的状态无限地增长。这个简单的网络在（1，1）有一个固定点吸引子：

$$\lim_{t=\infty} A(t) = 1$$

$$\lim_{t=\infty} \mathrm{DNA}(t) = 1$$

更一般地说，线性表达网络是影响网络，矩阵 $\boldsymbol{W}$ 是从外出链路加权获得的矩阵邻接，向量 $S(t)$ 是对应于节点值的状态向量。由于加权表达链路的变化率是由状态方程给出：

$$\frac{\partial S}{\partial t} = \boldsymbol{W}^{\mathrm{T}} S; S(0) = S_0$$

$$\boldsymbol{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \cdots & \cdots & \cdots & \cdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,n} \end{bmatrix}$$

连接矩阵 $\boldsymbol{W}$ 的元素对于前反馈激活表达是正的，对于后反馈和抑制剂表达为负的。向量 $S(0)$ 是表达网络中节点的初始状态。

我们对网络状态方程的固定点解特别感兴趣，当然是如果该解存在的话。回顾在线性系统中，仅当 $\boldsymbol{W}^{\mathrm{T}}$ 的谱半径受限于零时，固定点才可能存在：

$$\rho(\boldsymbol{W}^{\mathrm{T}}) < 0$$

此外，在线性影响网络中，奇异吸引子值等于每个节点在消失点的状态：

$$\lim_{t=\infty} S(t) = [\boldsymbol{W}^{\mathrm{T}} + I]^t$$

程序 BioNet. jar 通过迭代矩阵乘法 t^* 次直到 abs$[(S(t+1) - S(t))/S(t)]$ 几乎没有变化为止来近似吸引子。但是，这不能保证一个任意蛋白表达网络将收敛到某个有限值的固定点上。事实上，最有可能的结果是不收敛！因此，我们必须检查稳定性。

线性生物网络的稳定性取决于谱半径，但这个理论只适用于线性系统。到目前为止，我们假设网络是线性系统，但这种假设在自然界中可能不成立。事实上，几乎没有什么数学工具可以用来评估一个非线性网络的稳定性。

虽然如此，我们可以研究线性系统的稳定性，并学习一些有关生物网络的性质。

考虑图 13-3 的简单线性表达网络。两个蛋白质节点 S_1 和 S_2，调节一个自动抑制 DNA 节点 S_3。我们会问："这个网络稳定吗？如果稳定，那是否会达到一个固定点？"什么是固定点？起初，$S_1(0) = 1.0$，$S_2(0) = 2.0$，$S_3(0) = 0$，权重 $w_{1,3} = 0.25$，$w_{2,3} = 0.50$，$w_{3,3} = -0.33$。所有其他的表达权重为零。

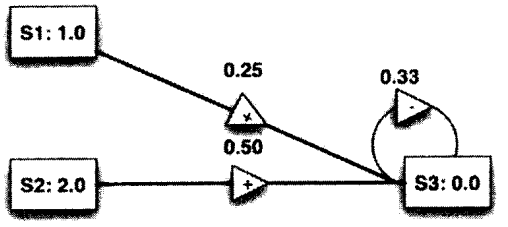


图 13-3 带有一个自动抑制节点并且表达权重分别为 1/4、1/2 和 1/3 的简单表达网络

在数学上，这种网络是由它的状态方程控制的：

$$\frac{\partial S}{\partial t} = \boldsymbol{W}^{\mathrm{T}} S; S(0) = \begin{bmatrix} 1.0 \\ 2.0 \\ 0.0 \end{bmatrix}$$

$$\boldsymbol{W} = \begin{bmatrix} 0 & 0 & 0.25 \\ 0 & 0 & 0.50 \\ 0 & 0 & -0.33 \end{bmatrix}$$

$$\rho(\boldsymbol{W}^{\mathrm{T}}) = -0.33$$

注意这个网络是稳定的，并且有一个固定点吸引子，因为它的谱半径小于零。事实上，没有振荡（虚）的部分，所以能够到达固定点而没有混沌振荡。这就能保证每个节点的吸引子的存在。对于初始状态 S_0，经过 $t^* = 20$ 次矩阵乘法，我们得到一个吸引子值：

$$[\boldsymbol{W}^{\mathrm{T}} + \boldsymbol{I}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.25 & 0.50 & 0.66 \end{bmatrix}^{20} \begin{bmatrix} 1.0 \\ 2.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.75 & 1.50 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1.0 \\ 2.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 2.0 \\ 3.75 \end{bmatrix}$$

吸引子分别是 1、2 和 3.75。程序 BioNet.jar 通过迭代找到近似解：1，2 和 3.79，因为 0.33 仅是 1/3 的一个近似。当初始状态为 $[2,5,0]^{\mathrm{T}}$ 时，节点 S_3 的吸引子值是多少？这留给读者作为练习。

13.2.2 布尔网络

线性连续模型具有其局限性，尤其是在建模非线性的下一个状态函数时更是如此。例如，不是每个节点的表达级别总和，而是假设级别是成倍增加。这种网络不再是线性的，所以就不能继续采用为影响网络所开发的线性分析工具进行分析了。事实上，没有数值分析技术，非线性连续模型就难以求解固定点值。非线性微分方程的数值积分是受不稳定性和可计算性限制的，许多研究人员已采取了简化方法进行近似。我们用同等的布尔网络取代连续网络。

*布尔网络*是一个包含逻辑运算 AND(*) 和 OR(+) 和布尔状态的生物网络。链路传输布尔信号：“+”代表逻辑上不变的信号，而“-”代表信号反转（如布尔 1 以 0 到达，而布尔 0 以 1 到达）。例如，一个负布尔反馈链路反置节点状态。

在生物学术语中，表达级别受到数字阈值限制：一个超过 0.5 的级别四舍五入为布尔 1，小于 0.5 的级别变成布尔零。同样，根据阈值 0.5，一个节点的状态既可以是 0 也可以是 1。在布尔网络中没有负值——只有 0 和 1。减法相当于一个比特位反转：1→0 或 0→1。

布尔表达网络本质上是数字电路。因此，我们的有机体模型近似于数字逻辑控制的机器模型。有机体的行为近似于逻辑电路，如图 13-4 中所示带有进位的*半加器*网络。这个“模体”可以在所有数字计算机中的算法单元中找到。它实现带进位的二进制加法——是所有计算机中将两个数加起来的最基本运算。例如，3 个带进位的半加器（并行）用于对两个二进制数求和，101 + 110：

+ 101

$\frac{+110}{+011}$ = 半加器

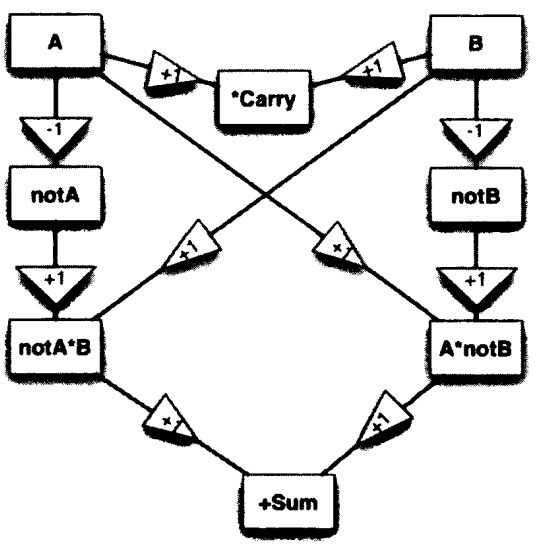
1000 = 进位

1011 = 11(十进制数字")

在图 13-4 中，设 $A=1$ 和 $B=0$ 代表两个数字（101 和 110）的最低有效位。进位很容易用乘法计算（AND)：1 * 0 = 0。半加器总和计算起来更复杂。首先，每位通过将节点 A 连接到 notA 以及将 B 连接到 notB 的链路置反。A 置反为 0，B 置反为 1，分别储存在节点 notA 和 notB 中。同时，A 和 B 比特位传送到乘数节点，notA * B 和 A * notB 。在节点 notA * B 乘积 0 * 0 = 0，在节点 A * notB 乘积 1 * 1 = 1。最后，OR 节点 Sum 将它的两个输入相加：0 + 1 = 1。半加位的最低有效位是 1。相似地，下一个有效位为 1，最高有效位是 0，进位为 1。十进制的总和是 11。

二进制加法模拟了计算机中的数字加法，但是在有机体上，AND 和 OR 逻辑近似于表达级别。例如，表达级别只在两个代谢产物在足够的浓度时才会“打开”——对应于逻辑 AND 节点。相反，如果只需一个或多个代谢产物激活反应时，使用 OR 节点。

布尔网络仅近似于线性连续网络。例如，当像布尔网络一样处理时，图 13-3 中的线性连续网络达到一个 S_3 的固定点，初始状态 $S_0=[1,1,0]^T$。这是很容易理解的。网络的第一次迭代，自动抑制剂链路置反 $S_3=0$ 为 $S_3=1$ 逻辑。从 S_2 布尔加 1 总和为 1，及从 S_1 添加 $\frac{1}{4}=0$ 都不改变总和。因此，S_3 以一步达到其布尔吸引子。



带进位的二进制加法器

图 13-4 带有进位的半加器网络包含 OR 和 AND 节点。该网络添加二进制数，每次一对比特

总的来讲，布尔网络的下一状态函数是对应于其布尔代数的逻辑表。在每一个时间步中，节点的下一个状态等于它的输入的逻辑组合。因此，对于图 13-3 的布尔代数方程组是：

$$S_1(t+1)=S_1(t);S_1(0)\text{ 已给出}$$

$$S_2(t+1)=S_2(t);S_2(0)\text{ 已给出}$$

$$S_3(t+1)=\lceil\lceil 0.5S_2(t)\rceil \text{OR} \lceil 0.25S_1(t)\rceil \text{OR}\{\text{NOT}\lceil 0.33S_3(t)\rceil\}\rceil;S_3(0)\text{ 已给出}$$

给定初始状态 $[1,2,0]^T$，这个网络的最终状态为：

$$S_1(t) = 1$$
$$S_2(t) = 2 = 1$$
$$S_3(t+1) = \lceil (1)\mathrm{OR}(0)\mathrm{OR}\{\mathrm{NOT}(0)\}\rceil = (1)\mathrm{OR}(0)\mathrm{OR}(1) = 1$$

假设初始状态为 $[1,0,0]^{\mathrm{T}}$。图 13-3 中网络的最终状态是什么？若不考虑时间，直接代入到上面的逻辑方程会给出误导的答案。正确答案的探究留给读者作为练习。

通过简化并将它作为一个布尔网络对待，我们就可以模拟非线性表达网络的行为。这使得我们可以直接应用布尔代数求得解，只不过它仍然是一种近似。这种方法也有分析的局限性，我们无法直接从布尔代数状态方程确定其稳定性或固定点吸引子值。

13.3 蛋白质表达网络

蛋白质表达网络是一个有向的影响网络，包含很多代谢物——基因、mRNA、DNA、蛋白质或其他分子——相当于节点，化学转换、翻译或转录表达以及其他调节关系相当于链路。每个节点状态通常是一些化学动态变化的浓度级别。链路的权重通常是一个表达级别的近似值或一种浓度对另外一种浓度的影响（一个节点对另一个节点的影响度）。正的链路代表一种表达的激活，负的链路代表一种表达的抑制。

布尔网络模型可以决定链路激活的存在（或不存在），以及节点上代谢物的存在（或不存在）。连续模型尝试着去预测一个有机体对摄动的响应。例如，当某一代谢物的浓度以一个确定的等级增长时会发生什么？除了传播规则不同外，通过整个网络的传播像传染病一样。

布尔网络的一般系统状态方程是一个逻辑方程，描述了每个节点从前一状态到下一状态及网络的连接拓扑。任意函数 $f(x)$ 可能是线性或非线性的：

$$S(t+1) = f(S(t)); S(0) = S_0$$

连续网络的一般系统状态是由微分方程组定义的，代表了浓度级别或质量的变化。同样，任意函数 $f(x)$ 可能是线性或非线性的：

$$\frac{\partial}{\partial t}S(t) = f(S(t)); S(0) = S_0$$

我们已经在前面的章节中了解到，由于网络中的前反馈连接，这种网络本来就不稳定并很容易“失效”。甚至线性网络模型也很可能是不稳定的，无法收敛到一个固定点解。通常，吸引子的增长值没有上限。这部分地是由于存在的源节点和接收节点缺乏控制浓度级别的反馈回路所造成的。还有部分原因是拓扑结构或网络的布线图造成的。我们可以通过谱分解更详细地探讨这个问题。

在线性网络中，我们知道稳定性取决于网络的谱半径。假如谱半径小于 0，网络实现其有限的固定点及吸引子是有界的。通常吸引子等于 0，经过一些振荡或抑制后节点到达 0。然而，假如谱半径超过 0，就不能保证有界。这就产生了有关生物网络和进化的问题。生物网络能否通过最小化谱半径的涌现过程而演变成随机网络？我们将在下一节中检验这一假设。

生物网络的涌现

蛋白表达网络用公式表示具有内在的不稳定性，因为它的谱半径倾向于大于 0——除非大量的链路是抑制剂因素，或者对链路权重有充分的限制。这主要是由于来自前反馈模体中正的有限邻接矩阵所导致的。此外，研究文献证实了生物网络往往是无标度的。这个证据产生了一个我们由模拟要解决的问题——生物在具有较低谱半径的无标度网络中是如何涌现的？

我们通过模拟测试两个假设：

假设1：从随机网络中涌现的带有高度hub的无标度网络，降低了生物网络的谱半径及促进了稳定性，因此“进化”达到某一固定值的生物网络。

假设2：从随机网络中涌现的带有高熵的生物网络，降低了其谱半径及促进了稳定性。这些涌现网络动态地收敛到稳定吸引子。

我们从之前的章节中知道，无标度网络的谱半径比同等随机网络的大约高75%。例如，n = 100个节点和m =200条链路的随机网络具有大约为4.8的谱半径，而由BA生成过程生成的网络产生一谱半径约为7.25的无标度网络。在第7章中，我们将n =100、m =200时的等效随机网络转变为带有大的hub和谱半径约为12.0的近似无标度网络。因此，我们有一种直觉，感觉到无标度网络的谱半径位于等效随机网络的2~3倍之间。

但是如果目标是要将随机网络转变为较小谱半径的生物网络，会发生什么呢？例如，假设将涌现过程应用到一个如下的随机网络上。从本章所定义的随机生物网络开始，随机地交换链路的端点，除非网络的谱半径降低，否则就拒绝交换。无限重复这个过程，直到被用户停止为止。那么对谱半径、hub和熵的影响是什么？

为了测试“假设1”，我们将如下涌现过程应用到一个由n =50个节点和m =100条激活链路（权值等于1）组成的随机生物网络上。允许涌现过程运行至少3000次迭代，一般来讲就足以到达最终状态。

最小化谱半径

1. 无限地重复：
   a. 随机地选择链路L，随机地选择头部H，或尾节点T。忽略抑制循环。
   b. 随机地选择显著的节点V。
   c. 随机地将H或者T重联到V，小心避免重复链路和循环。
   d. 计算谱半径ρ。如果ρ大于先前的谱半径，恢复L到它以前的链路。否则设置网络谱半径为ρ。

这种涌现过程（在程序BioNet.jar中实现）用来平均每个随机网络运行10次产生的谱半径。除了测量涌现过程对谱半径的影响外，我们也测量了它对hub度和熵的影响（参见图13-5）。

谱半径从1.67的平均值开始减小到接近零，但这对hub度和熵几乎没有影响。事实上，当谱半径下降100%时，熵和hub度每个参数只有4%的改变。因此，假设1被否定。

为了测试“假设2”，通过随机重联网络增加熵——既可以从头部也可以从尾部，仅保留增加熵的重联链路。所以重联是有偏见的。但是是哪种方式的偏见？从图13-5中我们看到熵和谱半径改变约为29%，hub度变化35%，也就是说，hub度几乎与熵（百分率标度）按比例增加。这就否定了假设2。

起初，结果似乎与随机网络的定义相反。怎样才能提高随机网络的随机性？锚定随机网络生成过程是用来产生这些仿真的结果的，但这并不能解释这个悖论。记住，随机网络是一个非常庞大的可能网络集合中的一个实例。因此，每次产生的随机网络，就有可能（以非零概率）不是最大熵的网络，而是考虑所有可能的组合节点对所有可能产生的网络分布中的一个样本。用于低估假定2的最大熵涌现过程，在可能的地方就会增加熵。

假设2被否定，是因为随着熵的最大化，谱半径会增加而不是减少。因此，我们必须抛弃稳定涌现及收敛的生物网络是最大和最小熵的结果的想法。然而随着hub度增加，这意味着它可以通过重联将随机网络变成类似于无标度的网络。这个未经证明的推测留给读者作为练习。大多数情况下，一些其他自然力量也会使生物网络进化为稳定、收敛的网络。

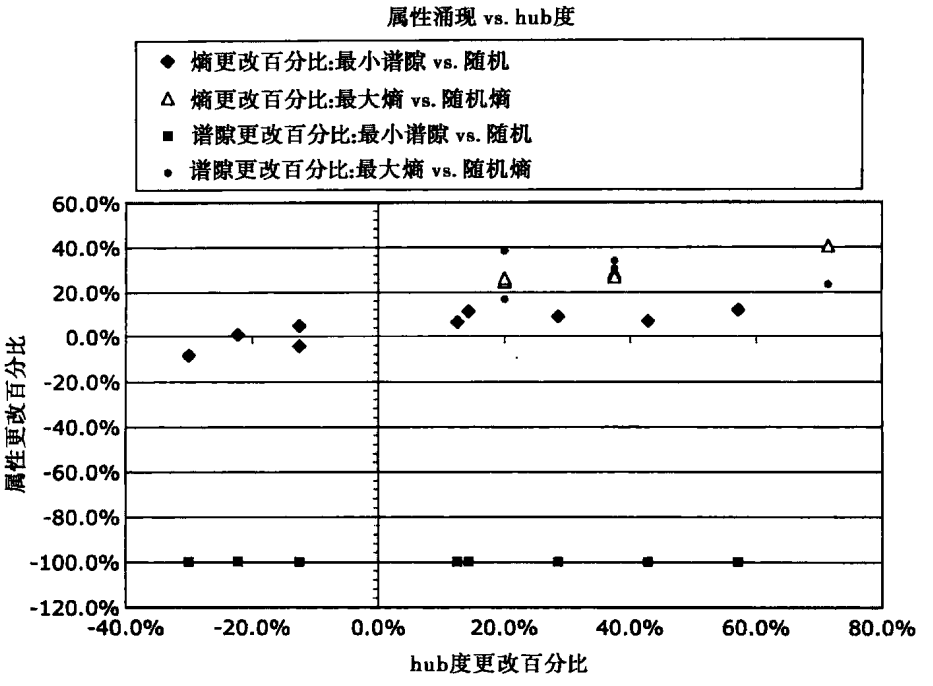


图 13-5 谱半径最小化和熵最大化涌现的模拟数据

13.4 质量动力学建模

上文中“失败的实验”表明稳定生物网络的简单涌现是不足以解释生物网络是如何从某一任意初始状态“计算出一个结束状态的”。它使我们怀疑到目前为止所开发的模型的有效性。在本节中，我们对有关已知生物网络动态特性的文献做进一步的扩展。

这个扩展是建立在合理的前提下——生物网络如以上描述的蛋白质表达网络本质上是质量动力学网络。*质量动力学*是一个化学分支，也就是在一种化学反应中反应剂浓度改变速率的模型。这在很大程度上关系到流过一个系统的液体。例如，质量动力学在细胞级别用于对代谢物浓度建模——为代谢网络提供一个状态方程式。这要求我们在每个网络节点服从*物质守恒定律*。节点状态转换速率如下：

$$\frac{\partial S}{\partial t} = \sum_{j \in \text{inlinks}} \text{inFlow}_j - \sum_{k \in \text{outlinks}} \text{outFlow}_k$$

其中 S =(流入)-(流出)。这种关系类似于*基尔霍夫定律*，但它不要求流出等于流入，因为节点可累积质量，而基尔霍夫节点的输入和输出必须平衡。这是当更改质量动力学网络行为时要注意的一个重要区别。

此外，节点的状态不能是负的：$0 \leqslant S(t)$ 。

该模型只适用于线性系统。然而，如果我们假设是线性的，那么较早开发的线性系统理论仍然适用，网络收敛的基本条件——以及固定点或吸引子值——就可以被近似。

13.4.1 质量动力学状态方程

考虑图 13-6a 中的简单表达网络。两个源节点 S_1 和 S_2 向一个接收节点 S_3 提供输入，但是以不同的级别，分别是25%和50%。以来自 S_1 、S_2 的输入和来自 S_3 本身的抑制反馈所确定的速率，节点 S_3 经历一次状态变化。质量动力学方程从这三个节点中建立了网络的状态方程：

$$\frac{\partial S_1}{\partial t} = -0.25S_1; S_1(0) = 1$$

$$\frac{\partial S_2}{\partial t} = -0.50S_2; S_2(0) = 2$$

$$\frac{\partial S_3}{\partial t} = 0.25S_1 + 0.50S_2 - 0.33S_3; S_3(0) = 0$$

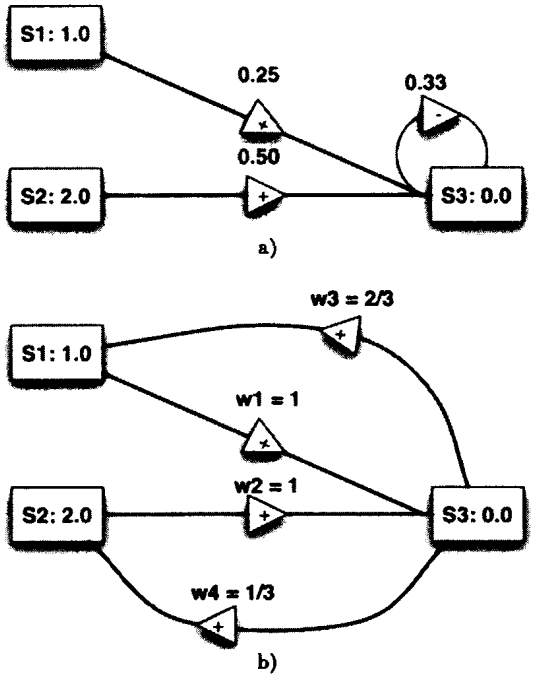


图 13-6 带有初始条件的质量动力学表达网络：a）类似于图 13-3 的简单网络；b）带有出度总和等于 1 的简单反馈网络

以矩阵表示成：

$$\frac{\partial S}{\partial t} = QS$$

$$Q = \begin{bmatrix} -0.25 & 0 & 0 \\ 0 & -0.50 & 0 \\ 0.25 & 0.50 & -0.67 \end{bmatrix} S(0) = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$$

这个系统有一个负的谱半径，$\rho(Q) = -0.25$，所以应该找到一个有限固定点的解而不会失效。事实上，所有的节点收敛到零，但在此之前按照图 13-7 中的曲线响应。这些响应曲线表明，由于抑制剂环路的抑制效应和 S_1、S_2 状态的损耗，这个简单的网络收敛到零。节点 S_1 和 S_2 分别从初始值 1 和 2 以指数方式下降，节点 S_3 从 0 急剧上升到 1.25，然后再下降到零。

图 13-6a 中质量动力学网络是稳定的，随着时间的推移在节点 S_1、S_2 抑制摄动。程序 BioNet. jar 可用来演示在图 13-7 中所示的响应，并在给定初始条件下估计出吸引子的值。这个例子支持动态网络的线性系统理论。但是如何确定矩阵 Q 呢？

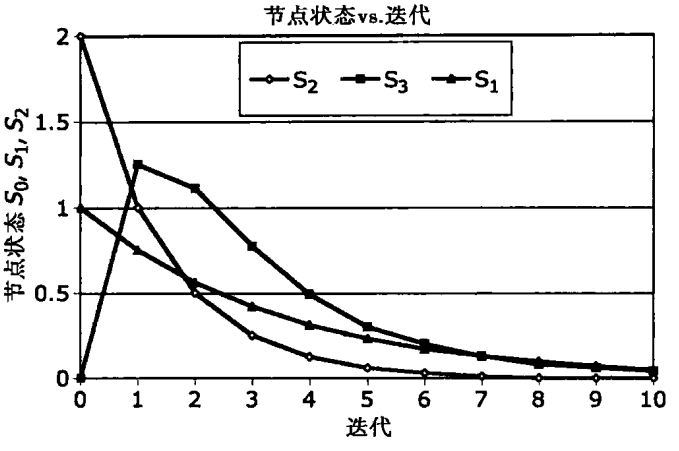


图 13-7 图 13-6a 中质量动力学网络的响应曲线

状态方程的输入部分是 $\boldsymbol{W}^{\mathrm{T}}\boldsymbol{S}$，$\boldsymbol{W}^{\mathrm{T}}$ 是转置邻接矩阵。以图 13-6 中的例子来说，邻接矩阵 $\boldsymbol{W}$ 和它的转置 $\boldsymbol{W}^{\mathrm{T}}$ 是：

$$\boldsymbol{W} = \begin{bmatrix} 0 & 0 & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} \\ 0 & 0 & -\frac{1}{3} \end{bmatrix}$$

$$\boldsymbol{W}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & -\frac{1}{3} \end{bmatrix}$$

状态方程的输出部分等于 $\boldsymbol{W}^{\mathrm{T}}$ 的 out_j 列的绝对值之和。这些列之和受到 100% 限制以维护质量守恒。我们从 $\boldsymbol{W}^{\mathrm{T}}$ 的对角减去 out_j 得到 $\boldsymbol{Q}$：

$$\mathrm{out}_j = \sum_i |w_{i,j}^{\mathrm{T}}|$$

$$\mathrm{OUT} = \begin{bmatrix} \sum_i |w_{i,1}^{\mathrm{T}}| & 0 & \cdots & 0 \\ 0 & \sum_i |w_{i,2}^{\mathrm{T}}| & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \sum_i |w_{i,n}^{\mathrm{T}}| \end{bmatrix}$$

$$Q = \boldsymbol{W}^{\mathrm{T}} - \mathrm{OUT}$$

$$\frac{\partial S}{\partial t} = QS(t) = [\boldsymbol{W}^{\mathrm{T}} - \mathrm{OUT}]S(t)$$

质量动力学网络的稳定性是由状态矩阵 $\boldsymbol{Q}$ 所决定的。如果 $\rho(\boldsymbol{Q}) < 0$，网络收敛到一个固定点或者重复包含在有限循环周期内的状态。在任何一种情况下，一个节点或更多的节点的状态可以振荡，而不是服从像图 13-7 中平滑的响应曲线。例如考虑到图 13-7 中的质量动力学网络，计算 $\boldsymbol{W}$，$\boldsymbol{W}^{\mathrm{T}}$，$\boldsymbol{Q}$，然后再计算 $\rho(\boldsymbol{Q})$：

$$\boldsymbol{W} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ \frac{2}{3} & \frac{1}{3} & 0 \end{bmatrix}$$

$$\boldsymbol{W}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & \frac{2}{3} \\ 0 & 0 & \frac{1}{3} \\ 1 & 1 & 0 \end{bmatrix}$$

$$\mathrm{OUT} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$\boldsymbol{Q} = \boldsymbol{W}^{\mathrm{T}} - \mathrm{OUT} = \begin{bmatrix} -1 & 0 & \frac{2}{3} \\ 0 & -1 & \frac{1}{3} \\ 1 & 1 & -1 \end{bmatrix}$$

因为$\rho(\boldsymbol{Q}) < 0$，这个网络收敛，但不是平滑地进行的。与寻找吸引子值并保持不变相反，每个节点在值间振动。这个网络是双稳定的，因为振荡既不增长也不降低。相反节点的状态落入有限的循环，无限重复下面的序列：

$$S_1: 0, 2, 0, 2, \cdots$$
$$S_2: 0, 1, 0, 1, \cdots$$
$$S_3: 0, 3, 0, 3, \cdots$$

这种有限循环是一个拐点——在稳定固定点吸引子与会导致最后失效的不稳定的振荡之间的微妙分界线。如果我们对参数稍做修改，那么在限制循环的两边都有可能找到吸引子。例如，如果从S_3到S_1，S_2的输出权重总和小于1.0（例如，$w_3 = 0.6$，$w_4 = 0.30$），网络在$S_1 = 0.95$，$S_2 = 0.47$，$S_3 = 1.58$时，找到一个稳定的固定点。另一方面，如果输出的权重总和超过1.0，网络将会失效。例如，如果$w_3 = 0.75$和$w_4 = 0.35$，那么从S_3的输出权重的总和为1.1。

13.4.2 有界的质量动力学网络

在生物学中，表达网络是一种相对较新的、未探索过的蛋白质相互作用的模型。质量动力学和网络模型之间的关系就更欠探索了。但是或许我们可以通过推测网络稳定性在生物学中的作用，深入了解网络科学和代谢过程之间的可能联系。然而，告诫读者这只是一个纯粹的推测，还未得到生物学的证实。

我们知道动态网络同步并到达稳定固定点的能力除了与其他因素有关外还与它的谱半径有关[⊖]。如果谱半径是负的并且极限为零，该网络就找到一个有限的固定点。在某些情况下，我们需要保持更多的条件，同样如基尔霍夫网络也是如此。一般来讲，为了保证一个动态网络不会失效，谱半径必须小于零。但特别是当它针对上述描述的质量动力学网络时，是否会有更低的谱半径值的优点？具有更低谱半径的生物网络是否要比用具有更高谱半径的网络更“拟合”？

考虑以下带有$n = 50$个节点和$m = 100$条链路（无标度网络的$\Delta m = 2$）的随机和无标度的质量动力学网络。在每种情况下，使用带有加权的有向链路的ER或BA生成过程生成一个网络。

⊖ “其他”包括增强的反馈回路。

谱半径取决于链路的有向性和权重以及度序列。假设链路的方向决定于 ER 和 BA 生成过程。那么其余可变的因素就是权重与拓扑。因此，分析结果谱半径便作为一种权重和网络类的函数。

图 13-8 展示了使用程序 BioNet. jar 生成具有不同链路权重分配模式的随机和无标度网络的 10 个模拟结果。谱半径是平均的，并且绘制如图所示，显示了正负标准偏差值。在第一类中（权重 = 1），所有链路都被分配为 1 权重。第二类（OUT 和 = 1），所有链路都被分配了一个等于 1/node [tail]. out_degree 的值，其中 node[tail]. out_degree 是对应链路的锚定或尾部节点的出度。同样，第三类（IN 和 = 1）权重分配，使所有输入到节点的链路和等于 1。最后，在区间 [0，1）中的随机权重被分配给所有随机权重类的所有链路。

如果我们相信带有低谱半径的网络更稳定——因此而更加拟合，那么图 13-8 给出的排序如下：

无标度（权重 = 1）　　最稳定
无标度（OUT 和 = 1）
随机（OUT 和 = 1）
随机（权重 = 1）
随机（IN 和 = 1）
无标度（随机权重）
无标度（IN 和 = 1）
随机（随机权重）　　最不稳定

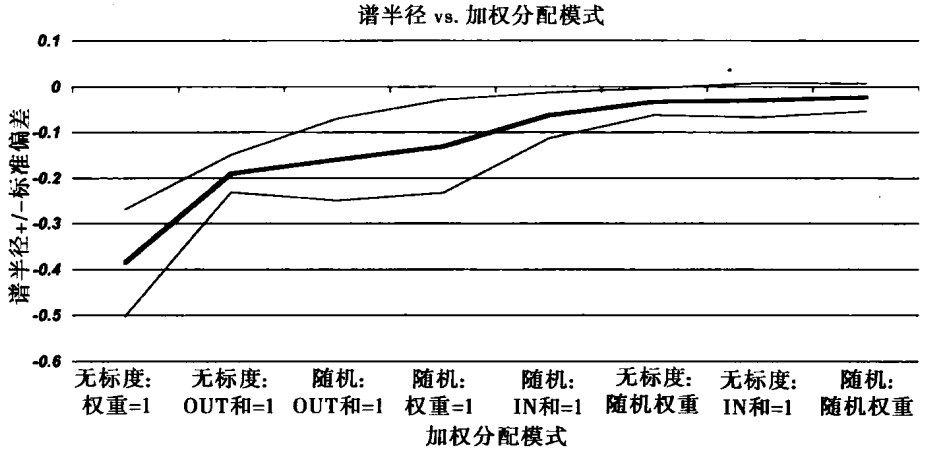


图 13-8　随机和无标度质量动力学网络的谱半径与链路权重分配类的对比

这些模拟结果不是有关无标度网络比随机网络优越的决定性结论的证据，因为当它们的输出链路权重总和等于 1 时，无标度和随机网络的谱半径差不多。这看起来似乎遵守物质守恒定律——通过从每个节点的输出限制为 100%，更关键的是降低谱半径而非度序列。3 个最低谱半径结果中的 2 个涉及无标度拓扑，2 个根据“OUT = 1”的约束。

网络科学与新陈代谢和蛋白质科学的组合是令人激动的新的研究领域，具有很多未解的问题。我们仅在本章进行了肤浅的介绍，这些问题的答案留给读者作为练习。

练习

13.1　当表达网络的初始状态为 $[2,5,0]^T$ 时，图 13-3 中节点 S_3 的吸引子值是多少？

13.2　当初始状态为 $[1,0,0]^T$ 时，图 13-3 中表达式网络的等价布尔网络的最终状态是什么？

13.3　如果被当做蛋白质表达网络时，图 13-3 中网络的所有特征值的值为多少？如果被当做质

量动力学网络对待时又如何呢?

13.4 图13-6b中的质量动力学网络在 $w_3 = \frac{3}{4}$、$w_4 = \frac{1}{4}$ 时稳定吗?如果稳定的话,网络的吸引子为多少?

13.5 为图13-1a中的双扇模体构造一个逻辑表,并证明当被当做布尔网络时相当于逻辑或门,也就是说输出 C、D 是通过简单的OR(或)操作获得的:$C = A$ or B,$D = A$ or B。

参考文献

Adamic, L. A., The small world web, in *Research and Advanced Technology for Digital Libraries, Lecture Notes in Computer Science 1696*, S. Abiteboul and A.-M. Vercoustre, eds, Springer-Verlag, New York, 1999, 443–452.

Adamic, L. A. and B. A. Huberman, Power-law distribution of the World Wide Web, *Science* **287**:2115 (2000).

Adamic, L. A., R. M. Lukose, A. R., Puniyani, and B. A. Huberman, Search in power-law networks, *Phys. Rev. E* **64**(4):046135 (2001).

Adamic, L. A., R. M. Lukose, and B. A. Huberman, *Local Search in Unstructured Networks*, Wiley-VCH, Verlag, Berlin, 2002.

Albert, R., H. Jeong, and A.-L. Barabasi, Diameter of the World Wide Web, *Nature* **401**:130–131 (1999).

Albert, R., H. Jeong, and A. L. Barabasi, The Internet's Achilles' heel: Error and attack tolerance of complex networks, *Nature* **406**:378–382 (2000).

Albert, R. and A.-L. Barabasi, Statistical mechanics of complex networks, *Rev. Mod. Phys.* **74**:47–97 (2002).

Albert, R., Scale-free networks in cell biology, *J. Cell Sci.* **118**:4947–4957 (The Company of Biologists) (2005).

Albert, R., Network inference, analysis, and modeling in systems biology, the plant cell online, www.aspb.org, *Am. Soc. Plant Biol.* 1–12 (Nov. 2007).

Al-Mannai, W. and T. Lewis, Minimizing network risk with application to critical infrastructure protection, *J. Inform. Warfare* **6**(2):52–68 (Aug. 2007).

Amaral, L. A., A. Scala, M. Barthelemy, and H. E. Stanley, Classes of small-world networks, *Proc. Natl. Acad. Sci. USA* **97**(21):11149–11152 (2000).

Network Science: Theory and Practice. By Ted G. Lewis

Copyright © 2009 John Wiley & Sons, Inc.

Atay, F. M., T. Biyikoglu, and J. Jost, Synchronization of networks with prescribed degree distributions, *IEEE Trans. Circuits Syst. I* **53**(1):92–98 (2006).

Bagnoli, F. and M. Bezzi, Small world effects in evolution, *Phys. Rev. E* **64**:021914 (2001).

Ball, F. G., D. Mollison, and G. Scalia-Tomba, Epidemics with two levels of mixing, *Ann. Appl. Prob.* **7**(1):46–89 (1997).

Barabasi, A.-L., R. Albert, and H. Jeong, Emergence of scaling in random networks, *Science* **286**:509–512 (1999).

Barabasi, A.-L., E. Ravasz, and T. Vicsek, Deterministic scale-free networks, *Physica A* **299**(3–4):559–564 (2001).

Barabasi, A.-L., *Linked: The New Science of Networks*, Perseus Publishing, Cambridge, MA, 2002.

Barabasi, A.-L., H. Jeong, Z. Neda, E. Ravasz, A. Schubert, and T. Vicsek, Evolution of the social network of scientific collaborations, *Physica A* **311**(3–4):590–614 (2002).

Barahona, M. and L. M. Pecora, Synchronization in small-world systems, *Phys. Rev. Lett.* **89**:054101 (2002).

Barnes, J. A., *Social Network*, Addison-Wesley, Reading, MA, 1972.

Barrat, A., Comment on Small-world networks: Evidence for a crossover picture, preprint 9903323 available from http://arxiv.org/abs/cond-mat/ (1999).

Barrat, A. and M. Weigt, On the properties of small-world network models, *Eur. Phys. J. B* **13**:547 (2000); available as *Condensed Matter/9903411.*

Barthelemy, M. and L. A. N. Amaral, Small-world networks: Evidence for a crossover picture, *Phys. Rev. Lett.* **82**:3180 (1999).

Bass, F. M., A new product growth for model consumer durables, *Manage. Sci.* **15**:215–227 (1969).

Bass, P. I. and F. M. Bass, *Diffusion of Technology Generations: A Model of Adoption and Repeat Sales*, Working Paper, Bass Economics Inc. (www.basseconomics.com), Frisco, TX, 2001.

Ben-Naim, E. and P. L. Krapivsky, Size of outbreaks near the epidemic threshold, *Phys. Rev. E* **69**:050901 (2004).

Bollobás, B., *Random Graphs*, Academic Press, New York, 1985.

Bollobas, B., O. Riordan, J. Spencer, and G. Tusnady, The degree sequence of a scale-free random graph process, *Random Struct. Algorithms* **18**(3):279–290 (2001).

Bonacich, P., Factoring and weighing approaches to clique identification, *J. Math. Sociol.* **2**:113–120 (1972).

Bristor, J. M. and M. J. Ryan, Applying the small world phenomenon to organizational buying, in M. Kochen, ed., *The Small World*, Ablex, Norwood, NJ, 1989, Chapter 4, pp. 87–99.

Broder, A., S. R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, Graph structure in the web, *Comput. Networks* **33**(1–6):309–320, (2000).

Burt, R. S., *Structural Holes: The Social Structure of Competition*, Harvard Univ. Press, Cambridge, MA, 1992.

Callon, M., The dynamics of techno-economic networks, in P. S. Rod Coombs and V. Walsh, eds, *Technological Change and Company Strategies: Economic and Sociological Perspectives*, Harcourt Brace Jovanovich, San Diego, 1992, pp. 72–102.

Chen, L.-C. and K. M. Carley, The impact of countermeasure propagation on the prevalence of computer viruses, *IEEE Trans. Syst. Man Cybernetics* (Part B: *Cybernetics*) 1–11 (2004).

Christensen, C. C., *The Innovator's Dilemma*, HarperCollins, 2003.

Coleman, J., *Foundations of Social Theory*, Belknap Press, 1990.

Collins, J. J., and C. C. Chow, It's a small world, *Nature* **393**:409–410 (1998).

Comellas, F., J. Ozon, and J. G. Peters, Deterministic small-world communication networks, *Inform. Process. Lett.*, **76**(1–2):83–90 (2000).

Das, R., M. Mitchell, and J. P. Crutchfield, A genetic algorithm discovers particle based computation in cellular automata, in Y. Davidor, H. P. Schwefel, and R. Manner eds, *Parallel Problem Solving in Nature, Lecture Notes in Computer Science*, Springer, Berlin, 1994, pp. 344–353.

Davidsen, J., H. Ebel, and S. Bornholdt, Emergence of a small world from local interactions: Modeling acquaintance networks, *Phys. Rev. Lett.* **88**(12):128701 (2002).

Dezso, Z. and A.-L. Barabasi, Halting viruses in scale-free networks, *Phys. Rev. E* **65**(5) (2002).

Dorogovtsev, S. N., J. F. F. Mendes, and A. N. Samukhin, Structure of growing networks with preferential linking, *Phys. Rev. Lett.* **85**(21):4633–4636 (2000).

Dorogovtsev, S. N., A. V. Goltsev, and J. F. F. Mendes, Pseudofractal scale-free web, *Phys. Rev. E* **65**(6):066122 (2002).

Dorogovtsev, S. N. and J. F. F. Mendes, Evolution of networks, *Adv. Phys.* **51**(4):1079–1187 (2002).

Dorogovtsev, S. N. and J. F. F. Mendes, *Evolution of Networks—from Biological Nets to the Internet and WWW*, Oxford Univ. Press, 2003.

Erdös, P. and A. Rényi, On the evolution of random graphs, *Publ. Math. Inst. Hungar. Acad. Sci.* **5**:17–61 (1960).

Erdos, P. and A. Renyi, On random graphs I, in *Selected Papers of Alfred Renyi*, Akademiai Kiadu, Budapest, Hungary, 1976 (1st publication in 1959), Vol. 2, pp. 308–315.

Erdos, P. and A. Renyi, On the evolution of random graphs, in *Selected Papers of Alfred Renyi*, Akademiai Kiadu, Budapest, Hungary, 1976 (1st publication in 1960), Vol. 2, pp. 482–525.

Eriksen, K. A. and M. Hornquist, Scale-free growing networks imply linear preferential attachment, *Phys. Rev. E* **65**(1):017102 (2002).

Euler, Leonhard (1707–1783), Koenigsberg Bridges Problem.

Faloutsos, M., P. Faloutsos, and C. Faloutsos, On power-law relationships of the Internet topology, *Proc. ACM SIGCOMM '99 Conf. Applications, Technologies, Architectures, and Protocols for Computer Communication in Cambridge, MA, USA*, ACM Press, New York, 1999, pp. 251–262.

Forman, C. and A. Goldfarb, *ICT Diffusion to Businesses*, http://www.andrew.cmu.edu/user/cforman/ICTDiffusion.pdf.

Foster, J., *From Simplistic to Complex Systems in Economics, Discussion Paper* 335, School of Economics, Univ. Queensland, St. Lucia QLD 4072, Australia, j.foster@economics.uq.edu.au, Oct. 2004.

Gabbay, M., The effects of nonlinear interactions and network structure in small group opinion dynamics, *Physica A* **378**:118–126 (www.elsevier.com/locate/physa) (2007).

Giaever, G., A. M. Chu, L. Ni, C. Connelly, L. Riles, S. Veronneau, S. Dow, A. Lucau-Danila, K. Anderson, B. Andre, et al., Functional profiling of the Saccharomyces ceravisiae genome, *Nature* **418**:387–391 (2002).

Gilbert, E. N., Random graphs, *Ann. Math. Stat.* **30**(4):1141–1144 (1959).

Girvan, M. and M. E. J. Newman, Community structure in social and biological networks, *Proc. Natl. Acad. Sci. USA* **99**(12):7821–7826 (2002).

Gleiss, P. M., P. F. Stadler, A. Wagner, and D. A. Fell, *Small Cycles in Small Worlds*, cond-mat/0009124, Working Paper 0-10-058, Santa Fe Institute, 2000.

Goh, K.-I., B. Kahng, and D. Kim, Spectra and eigenvectors of scale-free networks, *Phys. Rev. E* **64**(5):051903 (2001).

Granovetter, M. S., The strength of weak ties, *Am. J. Sociol.* **78**(6):1360–1380 (1973).

Guare, J., *Six Degrees of Separation: A Play*. Vintage, New York, 1990.

Hansen, M. T., The search-transfer problem: The role of weak ties in sharing knowledge across organization subunits, *Admin. Sci. Quart.* **44**:82–111 (1999).

Hayes, B., Graph theory in practice: Part I, *Am. Sci.* **88**(1):9–13 (2000).

Hayes, B., Graph theory in practice: Part II, *Am. Sci.* **88**(2):104–109 (2000).

Holland, J. H., *Emergence: From Chaos to Order*, Addison-Wesley, Reading, MA, 1998.

Holme, P. and B. J. Kim, Growing scale-free networks with tunable clustering, *Phys. Rev. E* **65**(2):026107 (2002).

Hong, H., M. Y. Choi, and B. J. Kim, Synchronization on small-world networks, *Phys. Rev. E* **65**:026139 (2002),

Huberman, B. A. and L. A. Adamic, Growth dynamics of the worldwide web, *Nature* **401**:131–132 (1999).

Hunter, J. and R. L. Shotland, Treating data collected by the small world method as a Markov process, *Social Forces* **52**:321–332 (1974).

Jeong, H., B. Tombor, R. Albert, Z. N. Oltvai, and A.-L. Barabasi, The large-scale organization of metabolic networks, *Nature* **407**:651–654 (2000).

Jeong, H., Z. Neda, and A.-L. Barabasi, Measuring preferential attachment for evolving networks, *Europhys. Lett.* **61**(4):567–572 (2003).

Jost, J. and M. P. Joy, Spectral properties and synchronization in coupled map lattices, *Phys. Rev. E* **65**(1):016201 (2002).

Jung, S., S. Kim, and B. Kahng, A geometric fractal growth model for scale-free networks, *Phys. Rev. E* **65**(6):056101 (2002).

Kephart, J. O. and S. R. White, Directed-graph epidemiological models of computer viruses, *Proc. IEEE Computer Society Symp. Research in Security and Privacy, May 20–22*, 1991, pp. 343–359.

Kermack, W. O. and A. G. McKendrick, A contribution to the mathematical theory of epidemics. *Proc. Roy. Soc. Lond. A* **115**:700–721 (1927).

Kim, B. J., C. N. Yoon, S. K. Han, and H. Jeong, Path finding strategies in scale-free networks, *Phys. Rev. E* **65**(2):027103, (2002).

Kleinberg, J. M., S. R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins, The Web as a graph: Measurements, models, and methods, in T. Asano, H. Imai, D. Lee, S. Nakano, and T. Tokuyama, eds, *Proc. 5th Annual Intnatl. Conf. Computing and Combinatorics*, Tokyo, *Lecture Notes in Computer Science*, Vol. 1627, Springer-Verlag, Berlin, 1999.

Kleinberg, J. M., Navigation in a small world, *Nature* **406**:845 (2000).

Kleinberg, J. M., The small-world phenomenon: An algorithm perspective, in STOC, *Proc. 32nd Annual ACM Symp. Theory of Computing*, Portland, OR, ACM Press, New York, 2000, pp. 163–170.

Kleinfeld, J., Six degrees of separation: Urban myth? *Psychol. Today* (2002).

Kogut, B. and G. Walker, *The Small World of Firm Ownership and Acquisitions in Germany from 1993 to 1997: The Durability of National Networks*, Wharton School, Reginald H. Jones Center Working Paper.

Korte, C. and S. Milgram, Acquaintance networks between racial groups: Application of the small-world method, *J. Personality Soc. Psychol.* **15**:101–108 (1970).

Krapivsky, P. L., S. Redner, and F. A. Leyvraz, Connectivity of growing random networks, *Phys. Rev. Lett.* **85**(21):4629–4632 (2000).

Kretzschmar, M. and M. Morris, Measures of concurrency in networks and the spread of infectious disease, *Math. Biosci.* **133**:165–96 (1996).

Kulkarni, R. V., E. Almaas, and D. Stroud, Evolutionary dynamics in the Bak-Sneppen model on small-world networks, available as *Condensed Matter/9905066* (1999).

Kulkarni, R. V., E. Almaas, and D. Stroud, Exact results and scaling properties of small-world networks, available as *Condensed Matter/9908216* (1999).

Kuramoto, Y., *Chemical Oscillations, Waves, Turbulence*, Springer-Verlag, 1984; republished by Dover Publications, 2003 (www.doverpublications.com).

Lago-Fernandez, L. F., R. Huerta, F. Corbacho, and J. Siguenza, Fast response and temporal coding on coherent oscillations in small-world networks, available as *Condensed Matter/9909379* (1999).

Laherrere, J. and D. Sornette, Stretched exponential distributions in nature and economy: "Fat tails with characteristic scales," *Eur. Phys. J. B* **2**:525–539 (1998).

Lai, Y.-C., A. E. Motter, and T. Nishikawa, *Attacks and Cascades in Complex Networks, Lecture Notes in Physics*, Vol. **650**, 2004, pp. 299–310. http://www.springerlink.com/.

Latora, V. and M. Marchiori, Efficient behavior of small-world networks, *Phys. Rev. Lett.* **87**(19):198701 (2001).

Latour, B., Technology is society made durable, in J. Law, ed., *A Sociology of Monsters*, Routledge, London, 1991.

Levene, M., T. Fenner, G. Loizou, and R. Wheeldon, A stochastic model for the evolution of the web, *Comput. Networks* **39**:277–287 (2002).

Lewis, T. G., *Critical Infrastructure Protection in Homeland Security: Defending a Networked Nation*, Wiley, Aoboken, NJ, 2006.

Li, X. and G. Chen, A local-world evolving network model, *Physica A* **328**:274–286 (2002).

Lin, N., The small world technique as a theory-construction tool, in M. Kochen, ed., *The Small World*, Ablex, Norwood, NJ, 1989, Chapter 2, pp. 231–238.

Liu, J., T. Zhou, and S. Zhang, Chaos synchronization between linearly coupled chaotic system, *Chaos Solut. Fractals* **144**:529–541 (2002).

Liu, J., General complex dynamical network models and its synchronization criterions, *Proc. 22nd Chinese Control Conf.*, Yichang, China, Aug. 10–14, 2003, pp. 380–384.

Liu, J., X. Yu, and G. Chen, Chaos synchronization of general complex dynamical networks, *Physica A* **334**:281–302 (2004).

Liu, J., X. Yu, G. Chen, and D. Cheng, Characterizing the synchronizability of small world dynamical networks, *IEEE Trans. Circuits Syst. I* **51** (2004).

Luczak, T., Phase transition phenomena in random discrete structures, *Discrete Math.* **136**(1–3):225–242 (1994).

Lundberg, C. C., Patterns of acquaintanceship in society and complex organization: A comparative study of the small world problem, *Pacific Sociol. Rev.* **18**:206–222 (1975).

Major, J., Advanced techniques for modeling terrorism risk, *J. Risk Finance* **4**(1) (Fall 2002).

Marchiori, M. and V. Latora, Harmony in the small-world, *Physica A* **285**(3–4):539–546 (2000).

Mathias, N. and V. Gopal, Small worlds: How and why, available as *Condensed Matter/0002076* (2000).

Mathias, N. and V. Gopai, Small worlds: How and why, *Phys. Rev. E* **63** (2001).

Medina, A., I. Matta, and J. Byers, On the origin of power laws in Internet topologies, *ACM Comput. Commun. Rev.* **30**(2):18–28 (2000).

Menezes M. Argollo de, C. F. Moukarzel, and T. J. P. Penna, First-order transition in small-world networks, *Europhys. Lett.* **50**(5):574 (2000).

Mihail, M., C. Gkantsidis, A. Saberi, and E. Zegura, *On the Semantics of Internet Topologies*. Technical Report GIT-CC-02-07, College of Computing, Georgia Institute of Technology, 2002.

Milgram, S., The small world problem, *Psychol. Today* **2**:60–67 (1967).

Mitzenmacher, M., A brief history of generative models for power law and lognormal distributions, *Internet Math.* **1**(2):226–251 (2004).

Molloy, M. and B. Reed, A critical point for random graphs with a given degree sequence, *Random Struct. Algorithms* **6**:161–180 (1995).

Monasson, R., Diffusion, localization and dispersion relations on "small-world" lattices, available as *Condensed Matter/9903347* (1999).

Montoya, J. M. and R. V. Solé, *Small World Patterns in Food Webs*, Santa Fe Institute Working Paper 00-10-059, 2000.

Moore, C. and M. E. J. Newman, Epidemics and percolation in small world networks, *Phys. Rev. E* **61**:5678–5682 (2000).

Moore, C. and M. E. J. Newman, *Exact Solution of Site and Bond Percolation on Small-World Networks*, Santa Fe Institute Working Paper 00-01-007, 2000.

Moukarzel, C. F. and M. Argollo de Menezes, Infinite characteristic length on small-world systems, available as *Condensed Matter/9905131* (1999).

Mukherjee, G. and S. S. Manna, Quasistatic scale-free networks, *Phys. Rev. E* **67**(1):012101 (2003).

National Research Council, Committee on Network Science for Future Army Applications, *Network Science*, Board of Army Science and Technology, National Academies Press, www.nap.edu, 2005.

Newman, M. E. J. and D. J. Watts, Scaling and percolation in the small world network model, *Phys. Rev. E* **60**(6):7332–7342 (1999).

Newmam, M. E. J., C. Moore, and D. J. Watts, Mean-field solution of the small world network model, available as *Condensed Matter/9909165* (1999).

Newman, M. E. J. and D. J. Watts, Scaling and percolation in the small-world network model, *Phys. Rev. E* **60**:7332–7342 (1999).

Newman, M. E. J. and D. J. Watts, Renormalization group analysis of the small world network model, available as *Condensed Matter/9903357* (1999).

Newman, M. E. J., C. Moore, and D. J. Watts, Mean-field solution of the small-world network model, *Phys. Rev. Lett.* **84**(14):3201–3204 (2000).

Newman, M. E. J., Models of the small world: A review, *J. Stat. Phys.* **101**:819–841 (2000).

Newman, M. E. J., Small worlds, the structure of social networks, *J. Stat. Phys.* **101**(3/4) (2000).

Newman, M. E. J., *The Structure of Scientific Collaboration Networks*, Santa Fe Institute Working Paper 00-07-037.

Newman, M. E. J., *Who Is the Best Connected Scientist? A Study of Scientific Co-Authorship Networks*, Santa Fe Institute Working Paper 00-12-064, 2000.

Newman, M. E. J., S. H. Strogatz, and D. J.Watts, Random graphs with arbitrary degree distributions and their applications, *Phys. Rev. E* **64**(2):026118 (2001).

Newman, M. E. J., The structure of scientific collaboration networks, *Proc. Natl. Acad. Sci. USA* **98**(2):404–409 (2001).

Newman, M. E. J., Clustering and preferential attachment in growing networks, *Phys. Rev. E* **64**(2):025102 (2001).

Newman, M. E. J., D. J. Watts, and S. H. Strogatz, Random graph models of social networks, *Proc. Natl. Acad. Sci. USA* **99**(Suppl. 1):2566–2572 (2002).

Newman, M. E. J., S. Forrest, and J. Balthrop, Email networks and the spread of computer viruses, *Phys. Rev. E* **66**(3):035101 (2002).

Newman, M. E. J., Mixing patterns in networks, *Phys. Rev. E* **67**:026126 (2003).

Newman, M. E. J., The structure and function of complex networks, *SIAM Rev. Soci. Industr. Appl. Math.* **45**(2):167–256 (2003).

Norton, J. H. and F. M. Bass, A diffusion theory model of adoption and substitution for successive generations of high-technology products, *Manage. Sci.* **33**(9):1069–1086 (Sept. 1987).

Pandurangan, G., P. Raghavan, and E. Upfal, Building low-diameter P2P networks, FOCS, *Proc. 42nd Annual Symp. Foundations of Computer Science,* Las Vegas, NV, IEEE Computer Society Press, Los Alamitos, CA, 2001, pp. 492–499.

Park, J., Evolving adaptive organizations, *Perspect. Business Innov.* **4**:59–64 (2000).

Pastor-Satorras, R. and A. Vespignani, Epidemic spreading in scale-free networks, *Phys. Rev. Lett.* **86**(14):3200–3203 (2001).

Paxson, V. and S. Floyd, Why we don't know how to simulate the internet, *Proc. 1997 Winter Simulation Conf., Atlanta, GA, ACM Press, New York*, 1997, pp. 1037–1044.

Pool, I. de Sola and M. Kochen, Contacts and influence, *Soc. Networks* **1**:1–48 (1978); reprinted in M. Kochen, ed., *The Small World*, Ablex, Norwood, NJ, 1989, Chapter 1, pp. 3–51.

Powell, R., *Defending Strategic Terrorists over the Long Run: A Basic Approach to Resource Allocation*, Institute of Governmental Studies, Univ. California, Berkeley, Paper WP2006-34, 2006.

Powers, M. R. and Z. Shen, *Colonel Blotto in the War on Terror: Implications for Event Frequency*, Fox School Working Paper, Temple Univ., 2005.

Price, D. J. de S., Networks of scientific papers, *Science* **149**:510–515 (1965).

Price, D. J. de S., A general theory of bibliometric and other cumulative advantage processes, *J. Am. Soc. Inform. Sci.* **27**:292–306 (1976).

Rapoport, A. and W. J. Horvath, A study of a large sociogram, *Behav. Sci.* **6**:280–285 (1961).

Redner, S., How popular is your paper? An empirical study of the citation distribution, *Eur. Phys. J. B* **4**(2):131–134 (1998).

Rosen, E., *The Anatomy of Buzz*, Doubleday-Random House, 2000.

Ryan, B. and N. C. Gross, The diffusion of hybrid seed corn in two Iowa communities, *Rural Sociol.* **8**:15–24 (1943).

Schumpeter, J. A., *Capitalism, Socialism and Democracy*, 1942.

Sedgewick, R. and P. Flajolet, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, Reading, MA, 1996.

Simon, H. A., On a class of skew distribution functions, *Biometrika* **42**(3/4):425–440 (1955).

Solomonoff, R. and A. Rapoport, Connectivity of random nets, *Bull. Math. Biophys.* **13**:107–117 (1951).

Standish, R. K., Complexity and emergence, *Complexity Internatl.* **9** (2001).

Stevenson, W. B., B. Davidson, I. Manev, and K. Walsh, The small world of the university: A classroom exercise in the study of networks, *Connections* **20**(2):23–33 (1997).

Stoneman, P., *The Economics of Technological Diffusion*, Blackwell Publishers, Oxford, UK, 2002.

Strogatz, S., *SYNC: The Emerging Science of Spontaneous Order*, Hyperion, 2003.

Tadic, B., Adaptive random walks on the class of web graphs, *Eur. Phys. J. B* **23**(2):221–228 (2001).

Tadic, B., Dynamics of directed graphs: The world-wide web, *Physica A* **293**(1–2):273–284 (2001).

Tadic, B., Growth and structure of the world-wide web: Towards realistic modeling, *Comput. Phys. Commun.* **147**(1–2):586–589 (2002).

Tadic, B., Temporal fractal structures: Origin of power laws in the world-wide web, *Physica A* **314**(1–4):278–283 (2002).

Travers, J. and S. Milgram, An experimental study of the small world problem. *Sociometry* **32**:425 (1967).

Vazquez, A., R. Pastor-Satorras, and A. Vespignani, Large-scale topological and dynamical properties of the internet, *Phys. Rev. E* **65**(6):066130 (2002).

Vazquez, A. and M. Weigt, Computational complexity arising from degree correlations in networks, *Phys. Rev. E* **67**(2):027101 (2003).

Virtanen, S., *Properties of Non-uniform Random Graph Models*, Helsinki Univ. Technology, Laboratory for Theoretical Computer Science, Report FIN-02015 HUT, lab@tcs.hut, 2003.

Volchenkov, D. and P. Blanchard, An algorithm generating random graphs with power law degree distributions, *Physica A* **315**(3–4):677–690 (2002).

Vukadinovic, D., P. Huang, and T. Erlebach, On the spectrum and structure of internet topology graphs, in H. Unger, T. Bohme, and A. Mikler, eds, *Proc. 2nd Innatl. Workshop on Innovative Internet Computing Systems (IICS 2002)*, Khlungsborn, Germany, *Lecture Notes in Computer Science*, Vol. 2346, Springer-Verlag, Berlin, 2002, pp. 83–95.

Wagner, A. and D. Fell, *The Small World inside Large Metabolic Networks*, Santa Fe Institute Working Paper 00-07-041, 2000.

Walsh, T., Search in a small world, IJCAI'99, *Proc. 16th Intnatl. Joint Conf. Artificial Intelligence*, Stockholm, Sweden, Morgan Kaufmann Publishers, San Francisco, 1999, Vol. 2, pp. 1172–1177.

Walsh, T., Search on high degree graphs, IJCAI'01, *Proc. 17th Intnatl. Joint Conf. Artificial Intelligence*, Seattle, Morgan Kaufmann Publishers, San Francisco, 2001, pp. 266–274.

Wang, X. and G. Chen, Pinning control of scale-free dynamical networks, *Physica A* **310**:521–531 (2002).

Wang, X. and G. Chen, Synchronization in small-world dynamical networks, *J. Bifurcation Chaos* **12**(1):187–192 (2002).

Wang, X. and G. Chen, Synchronization in scale-free dynamical networks: Robustness and fragility, *IEEE Trans. Circuits Syst. I* **49**:54–62 (2002).

Wang, Z., D. Chakrabarti, C. Wang, and C. Faloutsos, Epidemic spreading in real networks: an eigenvalue viewpoint, *Proc. 22nd Intnatl. Symp. Reliable Distributed Systems,* Oct. 6–18, 2003.

Wang, Z., *Net Product Diffusion and Industry Lifecycle*, Univ. Chicago, Dept. Economics, Oct. 5, 2003, economics.uchicago.edu/download/Diffusion.pdf.

Wasserman, S. and K. Faust, *Social Network Analysis*, Cambridge Univ. Press, 1994.

Watts, D. J. and S. H. Strogatz, Collective Dynamics of "small world" networks, *Nature* **393**:440–442 (1998).

Watts, D., Networks, dynamics, and the small-world phenomenon, *Am. J. Sociol.* **105**:2 493–527 (Sept 1999).

Watts, D. J., *Small Worlds*, Princeton Univ. Press, Princeton, NJ, 1999.

Waxman, B. M., Routing of multipoint connections, *IEEE J. Select. Areas Commun.* **6**(9):1617–1622 (1988).

Weigt, M., Dynamics of heuristic optimization algorithms on random graphs, *Eur. Phys. J. B* **28**(3):369–381 (2002).

Wellman, B., J. Salaff, and D. Dimitrova, Computer networks as social networks: Virtual community, computer-supported cooperative work and telework, *Ann. Rev. Sociol.* **22**:213–238 (1996).

White, H. C., Search parameters for the small world problem, *Soc. Forces* **49**:259–264 (1970).

Yano, S. and E. Delfs, *The New Lanchester Strategy*, Lanchester Press, Sunnyvale, CA, 1990.

Yook, S. H., H. Jeong, A.-L. Barabasi, and Y. Tu, *Phys. Rev. Lett.* **86**:5835 (2001).

Yule, G. U., A Mathematical theory of evolution based on the conclusions of Dr. J. C. Willis, *Phil. Trans. Roy. Soc. Lond. B* **213**:21–87 (1925).

Yung, V. J. An exploration in the small world networks, available as *Condensed Matter/ 0004214* (2000/2001).

Zegura, E. W., K. L. Calvert, and S. Bhattacharjee, How to model an internetwork, *IEEE Infocom: The 15th Annual Joint Conf. IEEE Computer and Communications Societies*, San Francisco, IEEE Computer Society Press, Los Alamitos, CA, 1996, Vol. 2, pp. 594–602.

Zegura, E. W., K. L. Calvert, and M. J. Donahoo, A quantitative comparison of graph-based models for Internet topology, *IEEE/ACM Trans. Networking* **5**(6):770–783 (1997).

Zhang, H., A. Goel, and R. Govindan, Using the small world model to improve Freenet performance, *Comput. Networks* **46**(4):555–574 (2002).

Zhu, X., M. Gerstein, and M. Snyder, Getting connected: Analysis and principles of biological networks, *Genes Devel.* **21**:1010–1024 (2007).

Zipf, G. K., *Human Behavior and the Principle of Least Effort*, Addison-Wesley Press, Cambridge, MA, 1949.