论文分类号 TP31　　单位代码 10183
密级 内部　　研究生学号 19906031

吉 林 大 学
硕 士 学 位 论 文

网络优化算法的设计实现与比较

Implementation and Comparison of Network Optimization Algorithms

作者姓名： 焦洋
专　　业： 计算机应用技术
导师姓名及职称： 孙吉贵 教授

论文起止年月：2000 年 7 月至 2002 年 5 月

提　　要

本文以实际“物流决策支持系统”项目为背景，介绍了网络的计算机存储表示，介绍了网络优化方面的几类典型问题和相关的算法以及对这些算法的一些改进，设计实现了多种最短路算法、多种最大流算法和最小费用流算法，分析、测试和比较了每类算法中各算法对实际问题的适应性及运行效率上的差别，为应用这些算法解决实际问题提供了参考数据。同时提供了一个生成实验网络的产生器，它生成的以标准形式表示的网络问题，可以用于测试相关算法实现。本文最后总结了建立该系统的过程中遇到的问题以及进一步的工作。

第一章 引言

1．1 网络优化的重要意义

我们生活在一个网络社会中，从某种意义上说，现代社会是一个由计算机信息网络、电话通信网络、运输服务网络、能源和物质分派网络等各种网络所组成的复杂的网络系统。网络优化就是研究如何有效地计划、管理和控制网络系统，使之发挥最大的社会和经济效益。

网络优化是运筹学中的一个经典和重要的分支，它已被广泛地应用在管理科学、计算机科学、信息论、控制论、物理、化学、生物学和心理学等各个领域，并取得了丰硕的成果。网络优化算法有些可以用经典的线性规划方法来解决，同时也存在一些专用的算法，具有更高的效率，尤其是网络的节点达到一定规模以后，用传统的线性规划方法解决是比较困难的。此外，这些专用算法本身也有各自的特点，效率也不尽相同。

网络优化研究的主要问题有：路径分析，即对最佳路径和最短路径的求解。资源分配，即为网络中的网线和结点寻找最近(这里的远近是按阻碍强度的大小来确定的)的中心(资源发散或汇集地)。连通分析，即寻找从某一结点或网线出发能够到达的全部结点或网线。流分析，流分析的问题主要是按照某种最优化标准(时间最少、费用最低、路程最短或运送量最大等)设计运送方案。

网络优化算法在解决经济和后勤等问题，例如制造工具的生产、通讯网络中包的路由选择，运输网络中的任务分配上是一类应用广泛的优化算法，是一种功能强大的工具[1]。

1．2 系统的开发背景及主要工作

本文作为我们研发的“物流决策支持系统”项目的一部分，从算法应用的实际实现出发，讨论网络的计算机存储表示，设计实现了多种最短路算法、最大流算法和最小费用流算法，包括求最短路径的 Dijkstra 算法、BellmanFord 算法，求解最大流问题与增广路的 Ford-Fulkerson 算法、最大容量增广路算法、Dinic 算法、Dinic 算法的改进算法、最短增广路算法、预流推进算法、最高标号预流推进算法和求解最小费用流问题的消圈算法、最小费用路算法、原始-对偶算法；分析、测试和比较了每类算法中各算法对实际问题的适应性及运行效率上的差别。我们把重点放在描述算法、算法的执行过程和测试结果的分析与比较，不严格叙述概念、结论和证明过程，相关概念请参

阅文献[2,3]。

1．3 网络优化的历史与现状

瑞士数学家欧拉（E Euler）在 1736 年发表了一篇题为“依据几何位置的解题方法”的论文，有效地解决了哥尼斯堡七桥难题，这是有记载的第一篇图论论文，欧拉被公认为图论的创始人。

直到 1936 年，匈牙利数学家 O Konig 发表了《有限图与无限图的理论》，这是图论的第一本专著，在这之间的 200 年里，总的来讲图论的发展是缓慢的。

20 世纪中期，电子计算机的发展以及离散的数学问题具有越来越重要的地位，使得作为提供离散数学模型的图论得以迅速发展，成为运筹学中十分活跃的重要分支。

国内在 20 世纪 50 年代中期开始了关于网络优化方面的研究，尤其是近年来这个研究方向正在逐渐引起人们的重视，但是目前还没有得到广泛认可的实际应用系统。

目前国外已经把图论的研究成果广泛的应用到实际中，比如电讯网络、交通控制以及生产管理等方面。国内在规划设计和交通运输等方面也有较多应用。

第二章 图与网络基础

在实际生活中，人们为了反映一些对象之间的关系，常常在纸上用点和线画出各种各样的示意图。比如数个城市之间的铁路交通图，用点表示城市，用点和点之间的联线代表这两个城市之间的铁路线，就反映了这几个城市之间的铁路分布情况；几个球队之间的比赛情况，也可以用图表示出来，在这里用不同的点分别代表各支球队，某两个队之间比赛过，就在这两个队所对应的点之间连一条线，这条线不过其它的点。可见，可以用由点及点与点的联线所构成的图，去反映实际生活中，某些对象之间的某个特定的关系。下面我们介绍一下有向图和无向图的基本概念。

2．1 有向图与网络的基本概念

由上面的介绍可知，一个图是由一些点及一些点之间的联线（不带箭头或带箭头）所组成的，为了区别起见，我们把图中两点之间带箭头的联线称为弧，不带箭头的联线称为边。

如果一个图 G 是由点及弧所构成的，则称为有向图，记为 G=（V，A），式中 V，A 分别表示 G 的点集合和弧集合。一条方向是从 v_i 指向 v_j 的弧记为 a_k=（v_i，v_j），称 v_i，v_j 为弧 a_k 的端点，其中 v_i 为尾，v_j 为头，并称 v_j 在 G 中与 v_i 相邻，弧 a_k 称为与顶点 v_i，v_j 关联，并称弧 a_k 为 v_i 的出弧，为 v_j 的入弧，如果某两条弧至少有一个公共端点，则称这两条弧在图 G 中相邻。

如果对有向图 G 中的每条弧赋予一个或多个实数，得到的有向图称为赋权有向图或有向网络，简称为网络，为了讨论方便，本文中对图和网络不作严格区分，因为任何图总是可以赋权的。

有向图中的途径是该图的一些顶点 i_1，i_2，…，i_r 和弧 a_1，a_2，…，a_{r-1} 所组成的子图，这些顶点与弧可以交错排列成点弧序列 i_1，a_1，i_2，a_2，…，a_{r-1}，i_r，其中 a_k=（i_k，i_{k+1}）（k=1，2，…，r-1）。如果该序列中的所有弧都指向同一方向，则该途径称为有向途径。

有向图中的路是该图的不包含重复顶点的途径。路中的弧可以分为两类：其中一类弧的方向与路的起点到终点的方向一致，称为路的前向弧；另一类弧的方向与路的起点到终点的方向不一致，称为路的后向弧或反向弧。前向弧的集合一般记为 P^+，反向弧的集合记为 P^-。有向图中的有向路是该图的不包含重复顶点的有向途径，即不包含反向弧的路。

有向图 G=（V，A）中，如果（i_1，i_r）∈A 或（i_r，i_1）∈A，则路 i_1，i_2，…，I_r 加上弧（i_1，i_r）或（i_r，i_1）组成的途径称为该有向图的圈，即圈是起点和终点重合且

不含其它重复顶点的途径。

对于有向图 G=（V，A）中的两个顶点，如果在图中至少存在一条路把他们连接起来，则称这两个顶点是连通的。如果图中任意两个顶点都是连通的，则称该图为连通的；否则被称为不连通的。如果有向图中从任意一个顶点出发，都存在至少一条有向路到达任意另一个顶点，则称该图为强连通的。

设 S，T 是节点集合 V 的一个划分（即 S，T? V；S，T?? ；S∪T=V；S∩T=? ），则称两个端点分别位于 S，T 的弧为一个割，记为[S，T]={(i,j)∈A|i∈S,j∈T}∪{(j,i)∈A|i∈S,j∈T}。

2．2 无向图与无向网络的基本概念

有时我们关心的只是两个顶点之间的联系，而不关心这种联系的方向性，这样就自然得到了无向图和无向网络的概念。

一个无向图 G 是由一个非空有限集合 V 和 V 中某些元素的无序对集合 E 构成的，即 G=（V，E）。其中 V 仍称为图 G 的顶点集或节点集，V 中的每一个元素 v_i 称为该图的一个顶点或节点；E 通常称为图 G 的边集合，E 中的每一个元素 e_k 称为该图的一条边。

边上赋权的无向图称为赋权无向图或无向网络。同有向图情况一样，本文对无向图和无向网络也不作严格区分。有向图的其他相关概念都可以自然地推广到无向图上来，当然也会有一些微小的差异：如无向图的一条边的端点不再有头和尾之分；无向图中的一条路上的边也不再有前向和反向之分；无向图的连通性不再有所谓的强连通概念；等等。

下面再介绍两种特殊的图。若无向图 G 的任意两个顶点间有且只有一条边相连，则称其为完全图。n 阶完全图记为 K_n。若图 G=（V，E）的顶点集 V 可以表示成两个互不相交的非空子集 S，T 的并集，且使 G 中的每一条边的一个端点在 S 中，另一端点在 T 中，则称 G 为二部图。当|S|=p，|T|=q，且 S 与 T 中任意两对顶点都相邻时，则称 G 为完全二部图，记为 $K_{p,q}$。类似地，也可以定义有向的完全图和有向的完全二部图。

其它与图相关的概念还有很多，其中有些概念我们会在后面用到时再作介绍。值得注意的是，目前有关图的术语目前并不完全统一，不同的作者可能用不同的术语表达同一个概念，也可能使用的术语相同，表达的概念却并不完全相同，因此应注意防止由此而可能引起的误解。此外，本文中在不引起混淆的情况下，有时将有向图和无向图都简称为图，具体所指应可从上下文中看出。

第三章 最短路算法的设计实现和比较

3．1 网络在计算机中的几种表示方法及比较

一般来说，图与网络算法的执行效率受三个因素的制约：(1）算法本身的复杂性，包括时间复杂性和空间复杂性。(2）网络在计算机中的表示方法，以及中间结果的操作方案等具体实现。如邻接矩阵或邻接表表示，中间结果需要多大的存储空间，如何与原网络进行数据交换等等。(3）开发语言＋运行环境支持。

目前实现的算法主要采用 C/C++/VC++语言实现，可以说 C/C++语言的执行效率是比较高的，VC++在开发用户界面上具有优秀的功能。语言的运行环境主要是操作系统的支持，目前有两大类：unix 和 Windows，一般说来程序在 unix 平台下要比 windows 平台下运行时间要短。

虽然其中的第（2)、(3）点不是算法核心部分，但就处理实际问题的效果而言，其对计算机的处理能力是有影响的。我们首先讨论图与网络的数据结构。在机器实现中，用来描述图与网络有以下几种常用的表示方法：邻接矩阵表示法，关联矩阵表示法，弧表表示法，邻接表表示法，星型表示法。为方便起见，以下将图与网络视为同一概念，即皆为加权有向图。

邻接矩阵表示法：将图以邻接矩阵的形式存储在计算机中。如果两点间有一条弧，则邻接矩阵中对应的元素为 1；否则为 0。如图 3.1 所示图的邻接矩阵表示为图 3.2 所示的矩阵。

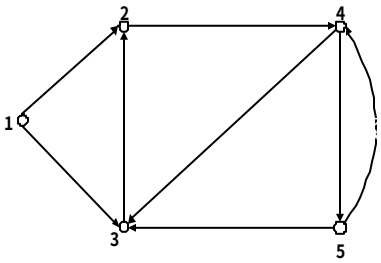


图 3.1

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

图 3.2

特点：表示简单、直接；如果网络比较稀疏，这种表示方法浪费大量的存储空间。矩阵在程序设计语言中以数组来表示，数组的申请和释放大都是静态的，操作方便，但对于网络的节点数大于预先申请的数组长度的情况不容易进行处理。弧上有多少种权，就需要有多少个邻接矩阵，应该说这么做对存储空间的浪费是比较大的。

对于加权图的邻接矩阵表示，一条弧所对应的元素不再是 1，而是相应的权值。对应权值代表的意义不同，“0”元具体填入何值可灵活处理。

从图 3.2 可以看出，本例非“0”元所占总的存储空间的比重为 32%，对于弧比较少即比较稀疏的网络，确实浪费了比较大的空间。

关联矩阵表示法：将图以关联矩阵的形式存储在计算机中。在关联矩阵中，每行对应于图的一个节点，每列对应于图的一条弧。如果一个节点是一条弧的起点，则关联矩阵中对应的元素为 1；如果一个节点是一条弧的终点，则关联矩阵中对应的元素为 -1；如果一个节点与一条弧不关联，则关联矩阵中对应的元素为 0。如图 3.1 所包含的弧为：(1，2)，(1，3)，(2，4)，(3，2)，(4，3)，(4，5)，(5，3)，(5，4)，则相应的关联矩阵表示为：

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 \end{bmatrix}$$

特点：关联矩阵中每一列的元素含有一个 1，一个－1，其他均为 0，如果网络中的弧赋有权，可以把关联矩阵增加一行，把每一条弧所对应的权存储在增加的行中。如果网络中的弧赋有多种权，可以把关联矩阵增加相应的行数，把每一条弧所对应的权存储在增加的行中。

可以看出，对于图 3.1，非“0”元所占总的存储空间的比重为 40%，由于无论网络含有多少个节点，其每列元素只有两个非“0”元，当图的节点数非常多的时候，浪费的存储空间仍是非常巨大的。

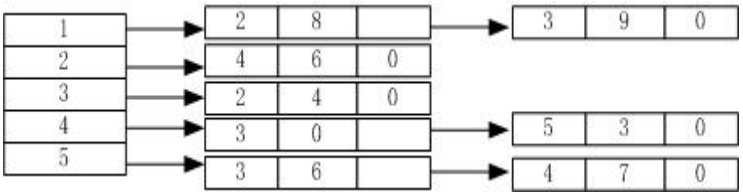
弧表表示法：将图以弧表的形式存储在计算机中。弧表就是图的弧集合中的所有有序对。弧表表示法直接列出所有弧的起点和终点，共需 2m 个存储单元，m 为弧的个数，此外，对于网络中每条弧上的权，也要对应地用额外的存储单元表示。假设图 3.1 中弧（1，2），(1，3)，(2，4)，(3，2)，(4，3)，(4，5)，(5，3)，(5，4）上的权分别为 8，9，6，4，0，3，6，7，则弧表表示如下：

起点	1	1	2	3	4	4	5	5
终点	2	3	4	2	3	5	3	4
权	8	9	6	4	0	3	6	7

特点：由于不存储无用的数据，存储空间达到很小，单从存储方面考虑弧表表示

法是比较有优势的，但寻找某一条弧的代价比较大。

邻接表表示法：将图以邻接表的形式存储在计算机中。所谓图的邻接表，就是图中所有节点的邻接表的集合，对每个节点，它的邻接表就是它的所有出弧。如图 3.1 的邻接表表示为：



特点：链表存储弧，空间上大体相当于弧表表示法，但由于弧的加入和删除涉及到内存“零散”的动态申请和释放，程序的正确性不容易把握，但同时可以看出，邻接表表示法具有非常好的灵活性，以及较小的存储空间，弧的寻找也比较简单，大多数程序存储网络结构都用邻接表。

星形表示法：把所有弧存放在一个数组中，在该数组中首先存放从节点 1 出发的所有弧，然后接着存放从节点 2 出发的所有弧，依次类推。为了能快速检索从每个节点出发所有弧，一般还用数组记录每个节点出发的弧的起始地址。如图 3.1 的星形表示为

节点对应的出弧的起始地址编号数组：（记为 point）

节点号 I	1	2	3	4	5	6
起始地址 point（I）	1	3	4	5	7	9

记录弧信息的数组

弧号	1	2	3	4	5	6	7	8
起点	1	1	2	3	4	4	5	5
终点	2	3	4	2	3	5	3	4
权	8	9	6	4	0	3	6	7

特点：

星形表示法主要分为前向星形表示法和反向星形表示法，对弧进行了编号，另外增加一个一维数组记录每个节点出发的弧的起始编号。前向星形表示法方便查找某个节点的出弧；反向星形表示法方便查找某个节点的入弧。星形表示法是弧表表示法的改进，由于对弧进行了编号，建立了节点在弧表中的初始地址，方便了弧的寻找。

一般在实际应用中，最常使用的是星形表示法和邻接表表示法。增加和删除一条弧，邻接表表示法所用的时间很少，而星形表示法所需工作量较大；邻接表表示法适用于提供指针类的语言，如 C，C++等，星形表示法则适用于不提供指针类的语言，如 Fortran 等。

3．2 最短路径算法的设计实现和比较

单源最短路径的问题描述：一个交通运输网络可以用一个加权有向图来描述，图的顶点表示城市，图中的各条弧表示城市之间的运输路线，每条弧上所赋的权值表示该路线的长度或沿此路线运输所花费的时间或运费等。这种运输路线往往具有方向性，例如汽车的上山和下山，轮船的顺水和逆水，所花费的时间或代价就不同。

所谓最短路径是指：如果从图中某一顶点（称为源点）到达另一顶点（称为终点）的路径可能不止一条，如何找到一条有向路径使得沿此路径上各弧的权值总和达到最小。

3.2.1 Dijkstra 算法

算法思想描述：按路径长度递增的次序，逐步产生最短路径的算法。首先求出长度最短的一条最短路径，然后参照它求出长度次短的一条最短路径，依次类推，直到从源点到其他各顶点的最短路径全部求出为止。该算法属于贪婪算法。

按照 Dijkstra 算法的思想求解图 3.3 所示网络，源点为节点 0，得到的 0 节点到其他各顶点的最短路径为： 0－>1,　0－>3－>2,　0－>3,　 0－>3－>2－>4。

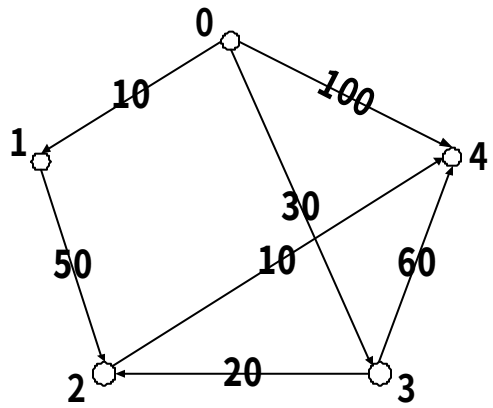


图 3.3

算法总的时间复杂度为 $O(n^2)$。邻接矩阵表示图，空间复杂度为 $O(n^2)$。

对于弧上含有负权值的情况，本算法有可能得不到正确的结果。如图 3.4 所示的例子，若节点 0 作为源点，按照 Dijkstra 算法其到 1，2 的最短路径分别为 7，5。显然这是不对的。

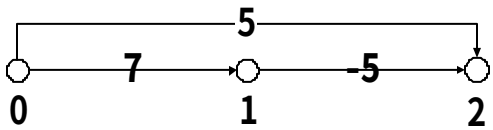


图 3.4

3.2.2Bellman-Ford 算法

算法思想描述：当网络中没有负圈时，有 n **个顶点的图中任意两个顶点之间如果存在最短路径，此路径最多有** n-1 **条弧。如果路径上的弧数超过了** n-1 **条时，必然会重复经过一个顶点，形成有向回路。**

BellmanFord **算法构造一个最短路径长度数组序列** $dist^1[u]$，$dist^2[u]$，…，$dist^{n-1}[u]$。**其中，**$dist^1[u]$**是从源点** v **到终点** u **的只经过一条弧的最短路径的长度，而** $dist^2[u]$**是从源点** v **最多经过两条边到达终点** u **的最短路径的长度，**…，$dist^{n-1}[u]$**是从源点** v **出发最多经过不构成负圈的** n-1 **条弧到达终点** u **的最短路径的长度。算法的最终目的是计算出** $dist^{n-1}[u]$。**例如初始网络如图** 3.5 **所示例子，其运行过程见表** 3.1。

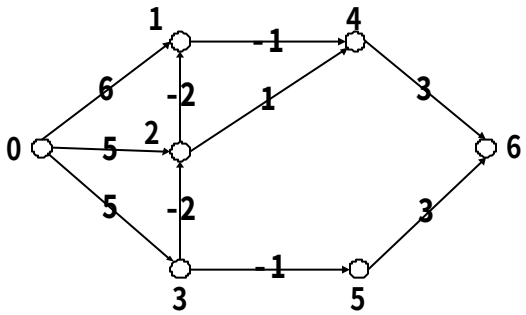


图 3.5

$dist^k$【u】**数组的变化**

K	$Dist^k[0]$	$Dist^k[1]$	$Dist^k[2]$	$Dist^k[3]$	$Dist^k[4]$	$Dist^k[5]$	$Dist^k[6]$
1	0	6	5	5	∞	∞	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

表 3.1

算法的时间复杂度为 $O(n^3)$。**邻接矩阵表示图，空间复杂度为** $O(n^2)$，**从实际问题出发，初始网络含有负圈不存在单源最短路径，本算法对这样的初始网络不能正确处理。**

3.2.3 两者的比较

Dijkstra 算法的复杂性比 BellmanFord 的复杂性要小，但没有 BellmanFord 算法的适应性广，可以根据实际问题选用不同的算法。BellmanFord 算法可以稍加改动形成在网络中寻找负圈的功能。另外，选用图的其他存储结构，如邻接表，可以在一定程度上降低算法的复杂度。

我们以三个实验样例[3]的运行具体说明 Dijkstra 算法和 BellmanFord 算法在执行效率上的差异。这三个实验样例分别记为 M_{10}、M_{30} 和 M_{50}，其规模如下：

M_{10}：102 个节点，290 条边，邻接矩阵大小 500？500。

M_{30}：902 个节点，2670 条边，邻接矩阵大小 3000？3000。

M_{50}：2502 个节点，7450 条边，邻接矩阵大小 7500？7500。

运行环境为 Windows2000，实现语言为 VC，P3 1000Hz CPU，256M RAM 运行结果如表 3.5：

	Dijkstra 算法（秒）	BellmanFord 算法（秒）
M_{10}	0.01	1.72
M_{30}	0.10	486.03
M_{50}	0.68	73719.14

表 3.5

就 Dijkstra 算法而言，若将邻接矩阵大小全部设为 7500？7500，则算法的执行时间全部在 0.68 秒－0.69 秒之间，这说明邻接矩阵表示网络除了浪费内存之外，还大量的增加了访问数组元素所花费的时间，而这种访问的时间代价在算法的整个运行时间上占有绝对的比重。因而采用邻接矩阵表示网络对于节点多的网络来说是非常不合适的。

第四章 最大流算法的设计实现和比较

4．1 最大流问题的数学描述

设 G=(P,A)加权有向图，L，U 分别为预先给定弧上权容量的下界和上界函数，L(ij)=l_{ij}，U(ij)=u_{ij}；D 为 G 的节点上的权函数，D(i)=d_i，表示节点 i 的供需量。此时称 G 为流网络，记为 G=（P,A,L,U,D)。

在流网络中，弧 ij 的容量下界 l_{ij} 和容量上界 u_{ij} 分别表示通过该弧发送某种“物质”时，必须发送的最小数量为 l_{ij}，而发送的最大数量为 u_{ij}。节点的供需量 d_i 则表示该节点从网络外部获得的“物质”的数量（d_i?0)，或者从该节点发送到网络外部的“物质”的数量（d_i?0)。当 d_i>0 时，节点 i 称为供应点或源；当 d_i<0 时，节点 i 称为需求点或汇；当 d_I=0 时，顶点 i 被称为转运点。

设 x 为流网络 G=（P, A, L, U, D）上的一个流，x 指派在弧 ij 上流量为 x_{ij}，如果 x 满足

$$\sum_{j:ij\in A} x_{ij} - \sum_{j:ji\in A} x_{ji} = d_i, \qquad i \in P$$

$$l_{ij} \le x_{ij} \le u_{ij} \qquad ij \in A$$

则称 x 为可行流。至少存在一个可行流的流网络称为可行网络。前一个等式称为流量守恒条件，后一个不等式称为容量约束。对于可行网络，必有 $\sum_{i\in P} d_i = 0$。

对于容量下界 L?0 的一般流网络情况总能将其转化为 L=0 的情况，因此这里仅考虑 L=0。另外，对于多源多汇的情况，这里不做一般讨论，仅讨论单源单汇这种特殊情形。设源为节点 s，汇为节点 t。则有流 x 的流量 v(x)=d_s= -d_t。记此时的流网络为 G=(s, t, P, A, U)。

最大流问题就是在 G=(s, t, P, A, U)中找到流值最大的 s-t 可行流，即最大流。在流网络 G=(s, t, P, A, U)中，对于流 x，如果对于任意 ij∈A，有 x_{ij}=0，则称 x 为零流，否则为非零流。若某条弧上的流量等于其容量，则称该弧为饱和弧；若某条弧上的流量为 0，则称该弧为空弧。

网络 G 中一条从 s 到 t 的路（未必是有向路）p 中，弧被分为两类：若弧 ij 的方向与路的起点 s 到终点 t 的方向一致，则称其为路 p 的前向弧；否则称为反向弧。P 的前向弧集和反向弧集分别记为 p+和 p-。

设 x 是流网络 G=（s，t，P，A，U）中给定的可行流，p 是一条 s－t 路，则 p 中满足下列两个条件之一的弧 ij 称为增广弧：

（1） 弧 ij 是 p 的前向弧且为不饱和弧；

（2） 弧 ij 是 p 的反向弧且为非空弧。

如果 p 中的弧都是增广弧，则称 p 为关于流 x 的增广路，简称增广路。

增广路定理 一个可行流为最大流的充要条件是不存在增广路。

最大流最小割定理 最大流的流值等于最小割的容量。

根据增广路定理，为了得到最大流，可以从任何一个可行流开始（如零流），沿增广路对流进行增广，直到网络中不存在增广路为止。

所有的增广路算法全部都是反复地找增广路进行增广，直到找不到增广路为止。不同算法之间的区别在于：

（1） 以多大的代价找到一条增广路。

（2） 找到的是一条什么样的增广路，或者说找到的增广路的效果如何。

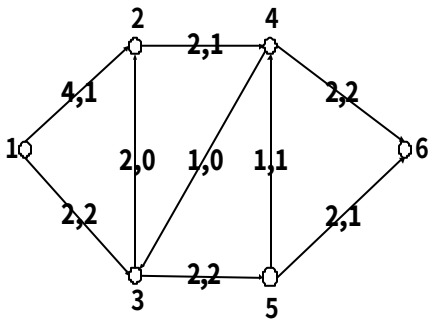
为了更有效地找到增广路，先考查网络关于流的残量网络：对网络 G＝（s，t，P，A，U）中给定的 s－t 可行流 x，网络 G 关于流 x 的残量网络 G（x）＝（s，t，P，A（x），U（x）），其中 A（x），U（x）定义如下：

$$A(x)=\{ij;\ ij?\ A,\ x_{ij}<u_{ij}\}?\ \{ji;\ \ ij?\ A,\ x_{ij}>0\},$$

$$U_{ij}(x)=\begin{cases}?u_{ij}?x_{ij}, & ij?A,\quad x_{ij}?u_{ij},\\ ?\\ ?\quad x_{ji}, & ji?A,\quad x_{ji}?0.\end{cases}$$

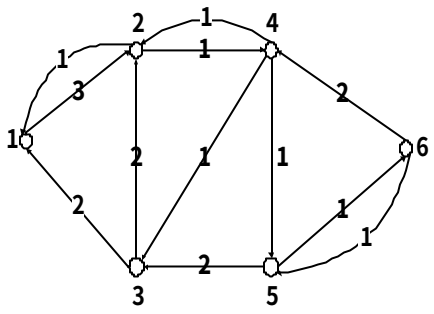
其中 u_{ij}（x）称为弧（i，j）上的残留容量。

举例说明，如图 4.1，原流网络中每条弧上的两个数字分别表示容量和当前流量：



流网络

图 4.1



对应的残量网络

图 4.2

残量网络如图 4.2，一条 s－t 不含零弧的有向路（以下简称有向路）就对应着原网络（图 4.1）中的一条增广路，因而在原网络中寻找增广路的过程就转化为在残量网络中寻找 s－t 有向路的过程。

4．2 几种最大流问题算法的实现和比较

下面叙述几种求增广路的算法。对于网络权值，可以是有理数，即计算机能够表示的有限小数的形式，为描述简单起见且不失一般性，以下例子的权值都取为整数。

4.2.1Ford-Fulkerson 算法

标准的 Ford-Fulkerson 算法是通过对节点进行标号的方法寻找增广路，本文 4.2 节中的最大容量增广路算法使用了标号的思想。在这里，我们叙述的 Ford-Fulkerson 算法是基于残量网络的。在残量网络中找到 s－t 路即可。

算法描述：

STEP0：置初始可行流。

STEP1：构造原网络的残量网络，在残量网络中找 s－t 有向路。如果没有，算法得到最大流结束。否则继续下一步。

STEP2：对应原网络中的 s－t 增广路，对于增广路中的前向弧，置 s(e) ？ u(e)- f(e)。对于反向弧，置 s(e) ？ f（e)。

STEP3：计算 crement=min{s（e_1)，s（e_2)，…，s（e_k）}；

STEP4：对于增广路中的前向弧，f(e) ？ f(e) ＋crement；对于反向弧，f(e)？ f(e)-crement，转 STEP1。

f 代表弧上的当前流量，s 表示弧上可增广的量。

在 STEP2 的残量网络中，寻找 s－t 有向路的算法有两种，DFS 和 BFS，即网络的深度优先和宽度优先遍历算法。

算法的时间复杂度为 O(mnU)，其中 m 为弧的数目，U 为弧上容量的最大上界，是伪多项式算法。邻接表表示图，空间复杂度为 O(n+m)。

DFS 和 BFS 的比较例子：

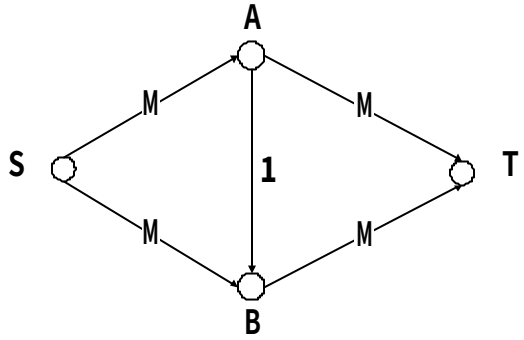


图 4.3

如图 4.3 所示，假设 M 是弧上的最大容量，且是一个非常大的整数，DFS 算法的最坏情况会选择 s－>a－>b－>t 和 s－>b－>a－>t 进行增广，最多增广的次数为 2M，这个问题用 BFS 算法，沿 s－>a－>t 和 s－>b－>t 增广两次就可以完成。

4.2.2 最大容量增广路算法

Ford-Fulkerson 算法每次只是在所有增广路中随机地找一条增广路进行增广，因此增广的次数可能很多。如果每次都找到一条增广容量最大的增广路，则总的增广次数应当减少，这样的算法称为最大容量增广路算法。最大容量增广路算法寻找增广路的思想是在遍历网络节点的过程中，记录到该节点为止可以增广的最大流量，此记录值称为标号，标号在原网络上进行，不是基于残量网络的。

算法描述：

STEP0：将 s 点可增广值 maxf 标记为一个非常大的数，其他节点的 maxf 值为 0，所有节点标记为未扩展。

STEP1：在所有有标记的节点中选择具有最大的 maxf（maxf>0）且还未扩展的节点 v1。若找不到任何一个这样的节点，此时已经没有增广路，结束本算法。若该节点是 t，增广路找到，结束本算法。否则继续下一步。

STEP2：从该节点出发，遍历以该节点为尾的弧 e=v1v2，将该节点标记为已扩展，若 maxf（v2）＝0，则令 maxf（v2）=min{maxf（v1），u（e）－f（e）}，否则，令 maxf（v2）=max{maxf（v2），min{maxf（v1），u（e）－f（e）}}，转 STEP1。

为了加快查找速度，节点的 maxf 值采用堆排序，本算法的实质是 maxf 值优先遍历。只需将每个节点 v 增加一个 maxf（v）值，表示通过该节点还可以继续增广的流量。在残量网络中，从源节点出发，进行 maxf 值优先遍历，直到遍历到汇节点为止。如此找到一条增广路。

例如图 4.4 给出的原网络(a)及算法执行过程图，其中节点旁的第一个数字表示节点编号，第二个数字表示该节点的 maxf 值，(a)中弧上的数字为该弧的容量：

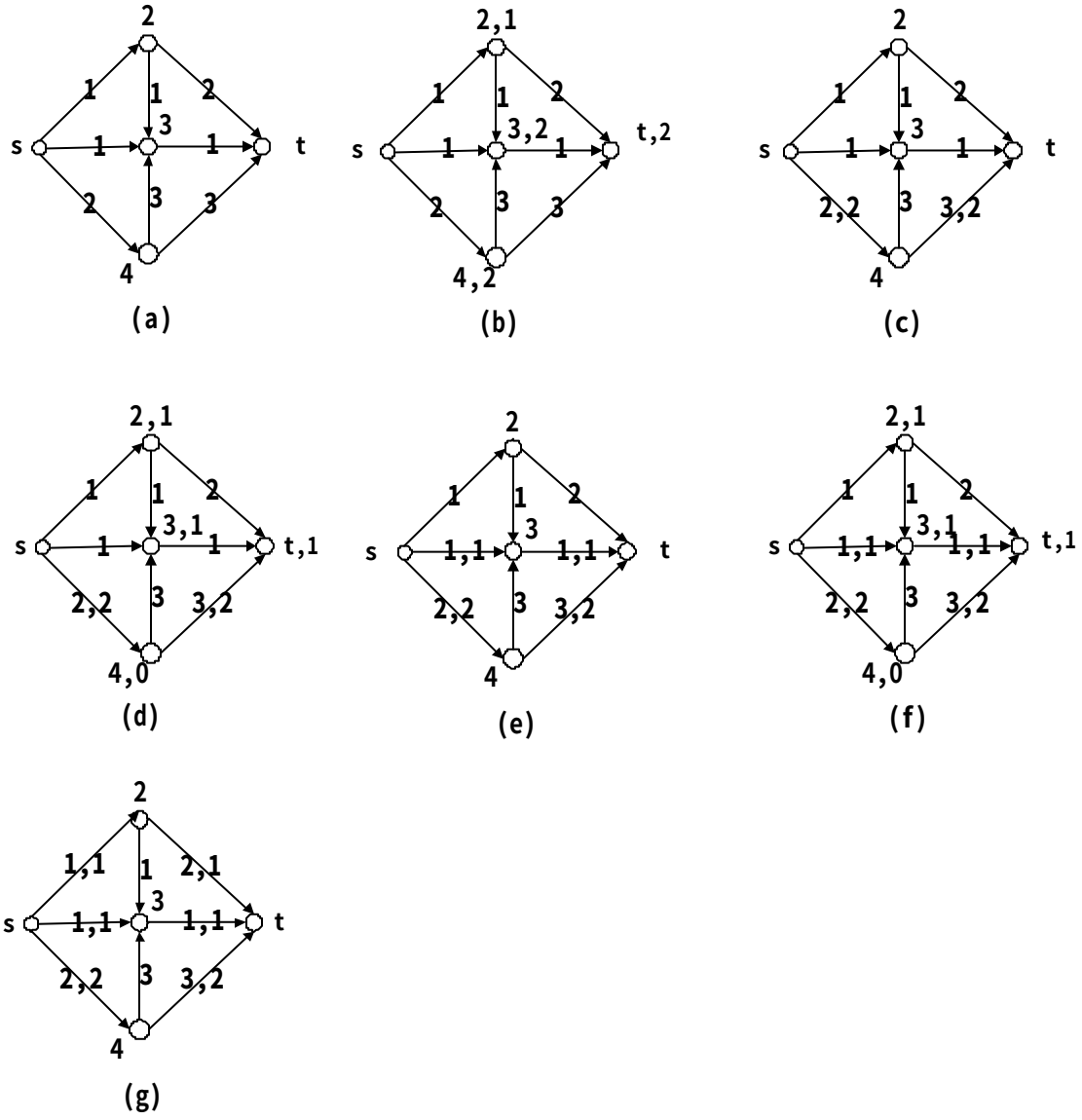


图 4.4

算法的时间复杂度为 O(mlogU)。其中 m 为弧的数目，U 为弧上容量的最大上界，是多项式算法。邻接表表示图，空间复杂度为 O(n+m)。

容量变尺度算法

最大容量增广路算法虽然降低了总的增广次数，但是每次都需要在残量网络中寻找最大容量增广路，因此每次增广的时间花费增加了。我们下面介绍的容量变尺度算法增广次数与最大容量增广路算法相同，而且每次在残量网络中寻找增广路的时间花

费也很低。

为了减少增广次数，容量变尺度算法每次都找一条增广容量“充分大”的增广路，但不一定最大。为此，我们构造一个残量网络的子网络Δ-残量网络，它是由容量不小于Δ的弧组成。

可见，对于整数 s-t 网络，在Δ-残量网络中查找到的增广路的流量至少为Δ，我们从Δ=$2^{?\log U?}$开始，每经过一定次数的增广后将Δ的值减少一半，直到Δ=1。

4.2.3Dinic 算法[5]

Dinic 算法的思想是为了减少增广次数，建立一个辅助网络 L，L 称为分层网络，L 与原网络 G 具有相同的节点数，但边上的容量有所不同，在 L 上进行增广，将增广后的流值回写到原网络上，再建立当前网络的辅助网络，如此反复，达到最大流。分层的目的是降低寻找增广路的代价。

算法描述：

STEP1：建造原网络 G 的一个分层网络 L。

STEP2：用增广路算法计算 L 的最大流 F，若在 L 中找不到增广路，算法结束。

SETP3：根据 F 更新 G 中的流 f，转 STEP1。

分层网络的构造算法：

STEP1：标号源节点 s，M[s]＝0。

STEP2：调用广度优先遍历算法，执行一步遍历操作，当前遍历的弧 e=v1－>v2，r＝G.u（e）－G.f（e)。

若 r>0 则

（1） 若 M[v2]还没有遍历，则 M[v2]=M[v1]+1，且将弧 e 加入到 L 中，容量 L.u（e）＝r。

（2） 若 M[v2]已经遍历且 M[v2]=M[v1]+1，且将边 e 加入到 L 中，容量 L.u（e）＝r。

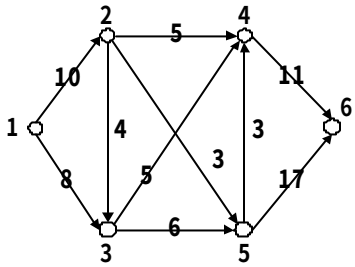
（3） 否则 L.u（e）＝0。

否则 L.c（e）＝0。

重复本步直至 G 遍历完。

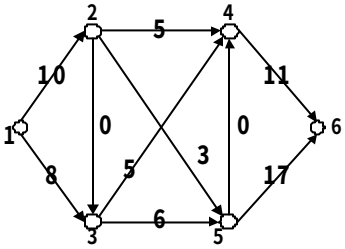
算法的时间复杂度为 O(mn^2)。其中 m 为弧的数目，是多项式算法。邻接表表示图，空间复杂度为 O(n+m)。

本算法的运行例子如图 4.5－4.11 所示：



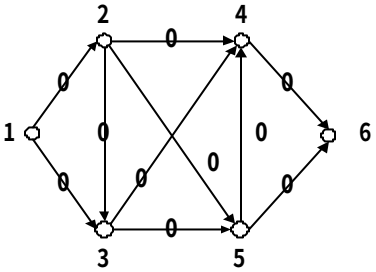
原始网络

图 4.5



分层网络

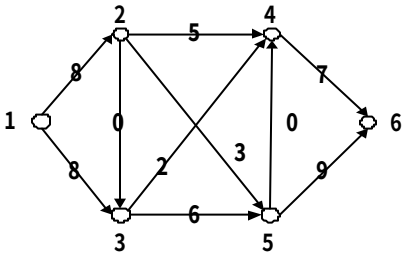
图 4.6



初始流网络

图 4.7

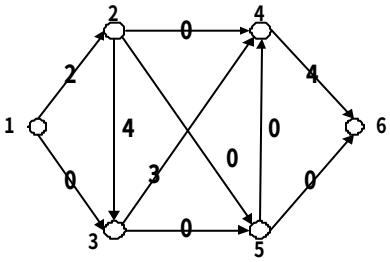
此时在分层网络上计算的最大流为图 4.8：



分层网络最大流

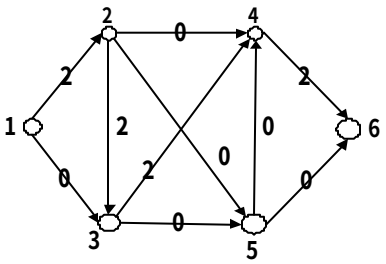
图 4.8

在当前流量为上图所示时，再构造分层网络及分层网络上的最大流为：



分层网络

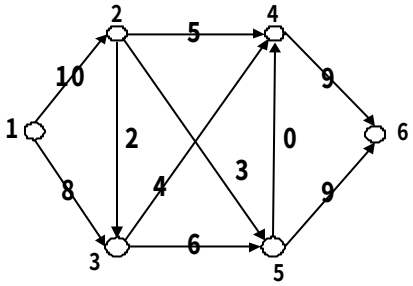
图 4.9



分层网络最大流

图 4.10

此时原网络的总的流量为：



总流量

图 4.11

此时再构造地分层网络已经无法继续增广，原网络 G 达到最大流。

Dinic 算法的改进算法

Dinic 算法具有比较高的效率，但由于要构造一个新的网络结构 L，并且要不断的把 L 中的流拷贝到原始网络 G 中，在这方面浪费了比较多的时间。Dinic 改进算法的思想是将 G 中的每一条边增加一个标志位，如果其上的容量有效，即置为 TRUE；否则置为 FALSE。置为 FALSE 的边不能参加增广，最大流是只包含 TRUE 边的最大流，其余同 Dinic 算法。改进后算法的复杂性没有改变，但执行效率提高得非常之多。

4.2.4 最短增广路算法[3]1

所谓最短路算法，即每次找到的都是一条包含弧数最少的增广路。

距离标号的定义：

对于一个残量网络 N（x），如果一个函数 d 将节点集合 V 映射到非负整数集合，则称 d 是关于残量网络 N（x）的距离函数，d（i）称为节点 i 的距离标号。如果距离函数满足：

（1） d（t）＝0，

（2） 对 N （x）中的任意一条弧（i，j）有 d（i）？d（j）＋1。

则称距离函数 d 关于流 x 是有效的，或称距离标号是有效的。如果任意一个节点的距离标号正好等于残量网络中从该节点到汇点（节点 t）的所有有向路中弧数最少的有向路所包含的弧数，则称距离函数 d 关于流 x 是精确的，或称距离标号是精确的。

假设残量网络如图 4.12 所示，节点 1 为源，5 为汇。则（a）中的标号是有效的，（b）中的标号是精确的。节点旁第一个数字为节点序号，第二个数字为节点距离标号。

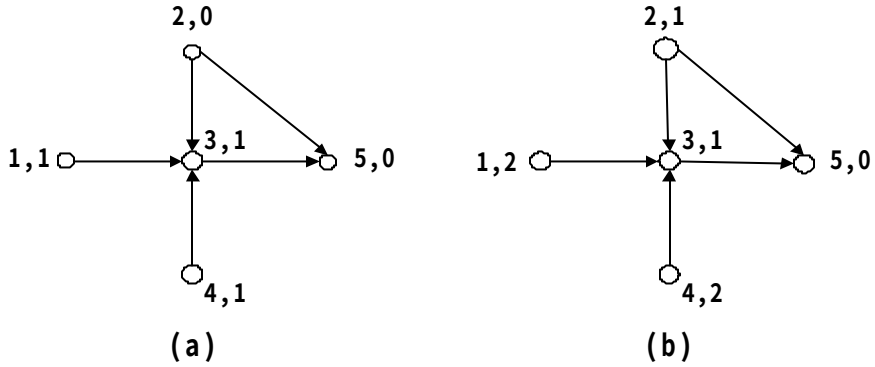


图 4.12

此外，如果对 N（x）中的某一条弧（i，j）有 d（i）＝d（j）＋1，则称弧（i，j）为允许弧。一条 s－t 有向路如果完全由允许弧组成，则该有向路称为允许路。

距离函数的性质：若距离函数是有效的，则：

（1） d（i）是残量网络 N（x）中从节点 i 到节点 t 的最短有向路长的下界。

（2） 如果 d（s）？n，则残量网络 N（x）中从节点 s 到节点 t 没有有向路（增广路）。

（3） 允许路是残量网络 N（x）中的最短增广路。

在一个残量网络中，对 N（x）沿反向弧进行广度优先搜索，即可以确定每个节点的精确距离标号。一个节点的精确的距离标号实际上表示的是从该节点到汇点（节点 t）的最短路路长，就是对所有节点按照最短路路长进行了层次划分。

最短增广路算法每次都找一条最短增广路，即残量网络 N（x）中的允许路进行增广。基于距离标号，具体算法可以描述如下：

STEP0：置初始可行流 x 为零流；计算精确的距离函数 d；令当前节点 i＝s。

STEP1：若 d（s）<n，继续下一步；否则结束，已经得到最优解 x。

STEP2：如果存在节点 i 的某条出弧（i，j）为允许弧，则转 STEP3；否则转 STEP4。

STEP3：令 pred（j）＝i，再令 i＝j；若 i＝t，则找到了一条增广路，进行增广，修改残量网络，并令当前节点 i＝s。转 STEP1。

STEP4：修改标号：当{（i，j）？A（x）|u_{ij}（x）>0}？？时，令 d（i）＝min{d（j）＋1|（i，j）？A（x）且 u_{ij}（x）>0}，否则令 d（i）＝n；且当 i？s 时再令 i＝pred（j）。转 STEP1。

最短增广路算法的运行例子如图 4.13 所示：

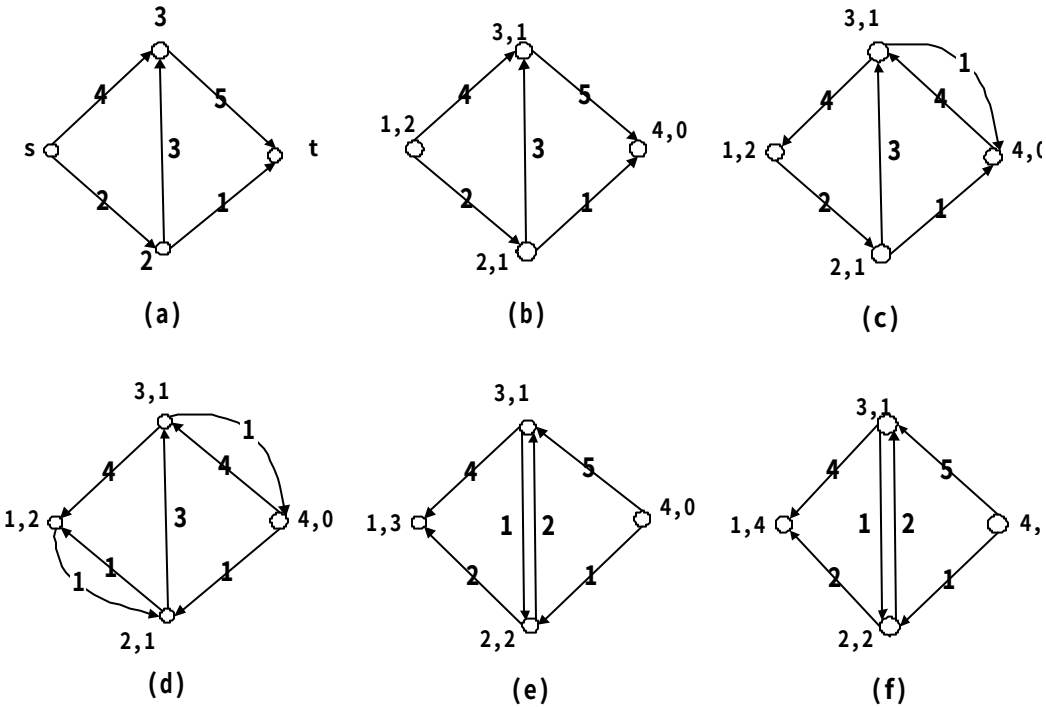


图 4.13

下面是几种增广路算法样例实际执行效率的比较：

	DFS（秒）	BFS（秒）	最大容量增广路（秒）	DINIC（秒）	DINIC_NEW（秒）	最短增广路（秒）
M10 增广次数	<0.01 286	<0.01 160	<0.01 53	<0.01 7	<0.01 7	<0.01 8
M30 增广次数	2.47 19553	0.15 1369	0.13 172	0.10 20	0.01 21	0.12 23
M50 增广次数	364.11 280109	6.87 3295	0.89 241	0.71 19	0.15 19	0.83 28

4.2.5 预流推进算法

我们简要分析增广路算法的不足：增广路算法的固有缺陷在于它找到增广路以后需要马上沿增广路对流进行增广，而每一次增广的复杂度为 O（n)。在某些情况下，这是极其浪费时间的。比如图 4.14 的例子。

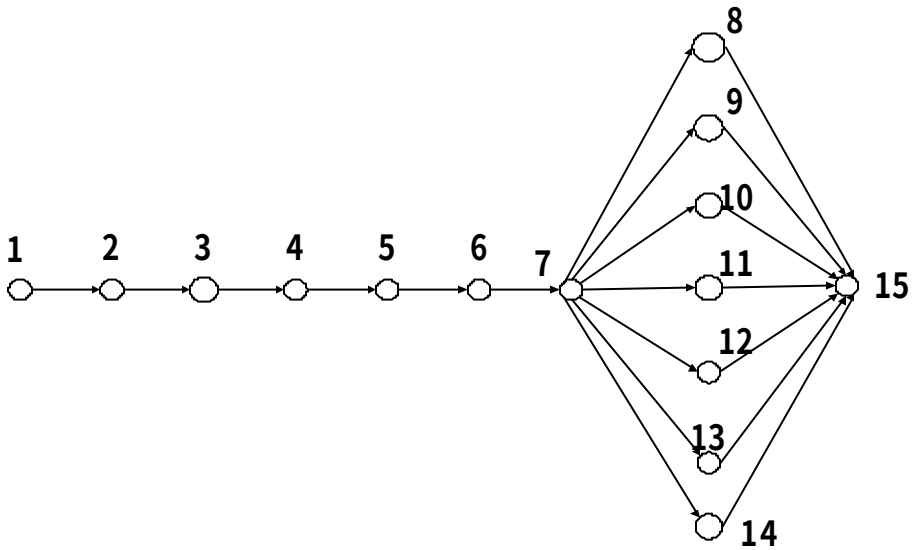


图 4.14

如上图所示，s＝1， t＝15。弧（1，2)，(2，3)，…（6，7）上的容量为 20，其他所有弧上的容量均为 1。无论采用上述提到的何种增广路算法，都会找到 7 条增广路，每条路长为

8，容量为 1。因此需要 7 次增广，每次增广 1 个单位，需要对 8 条弧进行操作。然而，8 条增广路中的前 7 个节点（前 6 条弧）是完全一样的，能否直接将前 6 条弧的流量增广 6 个单位，而只对后面长为 2 的不同的有向路单独操作呢？这样可以节省许多计算时间。这就是预流推进算法的思想。也就是说，预流推进算法关注于每一条弧的操作和处理，而不必一次一定处理一条增广路。

a.一般的预流推进算法[6]

预流的定义：流网络 N＝（s，t，V，A，U）上的一个预流 x 是指从 N 的弧集 A 到实数集合 R 的一个函数，使得对每个顶点 i 都满足如下条件：

$$e(i) \geq 0, \quad i \neq s, t,$$

$$0 \leq x_{ij} \leq u_{ij}, \ ? \ (i, j) \ ? \ A,$$

其中 $e(i) = \sum_{j:(j,i) \in A} x_{ji} - \sum_{j:(i,j) \in A} x_{ij}$ （? i ? V)，称为预流 x 在节点 i 上的赢余。e(i)>0

的节点 i（i？ s，t）称为活跃节点。

对流网络 N＝（s，t，V，A，U）上的一个预流 x，如果存在活跃节点，则说明该预流是不可行的。预流推进算法就是要选择活跃节点，并通过把一定的流量推进到它的邻居，近可能地将正的赢余减少为 0。如果当前活跃点有多个邻居，那么算法总是首先寻求把流量推进到它的邻居中距离节点 t 最近的节点。由于每个节点的距离标号可以表示其与节点 t 的距离，因此算法总是将流量沿着允许弧推进。如果当前活跃点出发没有允许弧，则增加该节点的距离标号，使得从当前活跃点出发至少含有一条允许弧。

算法描述：

STEP0：(预处理）置初始可行流 x 为零流；对节点 s 的每条出弧（s，j)，令 x_{sj} ＝ u_{sj} ；对任意的 i？V 计算精确的距离标号 d（i)；令 d（s）＝n。

STEP1：如果残量网络中不存在活跃节点，结束，已经得到最大流；否则继续。

STEP2：在网络中选取活跃节点 i；如果存在节点 i 的某条出弧（i，j）为允许弧，则将 min{e(i),u_{ij} (x)}个单位的流从节点 i 推进到节点 j；否则令 d（i）＝ min{d(j)+1|(i,j)？A(x)且 u_{ij} >0}.转 STEP1。

可见，算法的每一次迭代是一次推进操作或者一次重新标号操作。对于推进操作，如果推进的流量等于弧上的残留容量，则称为饱和推进，否则称为非饱和推进。当算法终止时，网络中不含有活跃节点，因此置有节点 s，t 的赢余为非零数。所以，此时得到的预流实际上已经是一个可行流。又由于在算法预处理阶段已经令距离标号 d（s）＝n，而距离标号在计算过程中不会减少，因此算法在计算过程中可以保证网络中永远不会有增广路存在。根据增广路定理，算法终止时一定得到了最大流。

在选择活跃节点的处理上主要有以下几种算法：

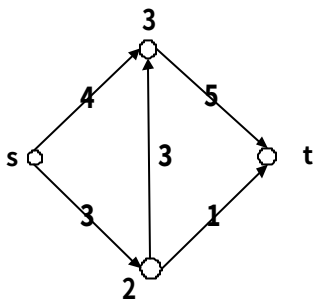
（1)、用队列来排列活跃节点

（2)、当前流量优先预流推进算法

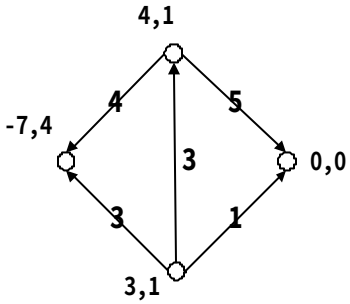
（3)、用栈来排列活跃节点

（4）广度优先预流推进算法

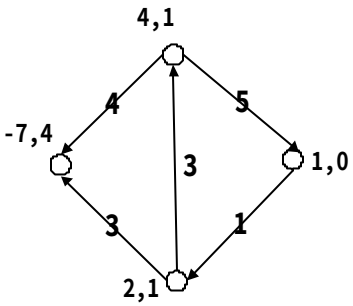
算法的运行例子如图 4.15 所示：


3
4
5
s
3
t
3
1
2

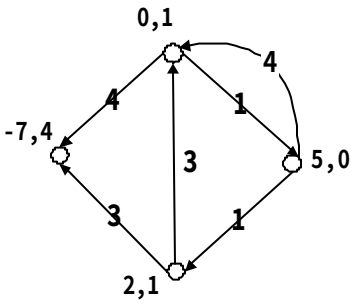
(a)


4,1
-7,4
4
5
3
0,0
3
1
3,1

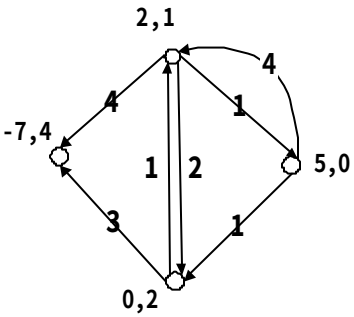
(b)


4,1
-7,4
4
5
3
1,0
3
1
2,1

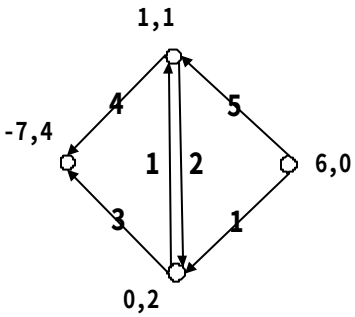
(c)


0,1
4
4
1
-7,4
3
5,0
3
1
2,1

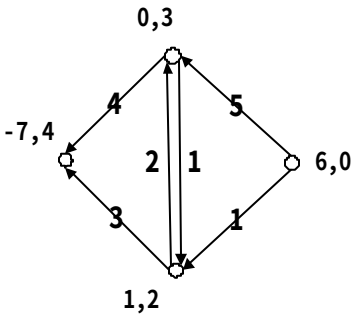
(d)


2,1
4
4
1
-7,4
1
2
5,0
3
1
0,2

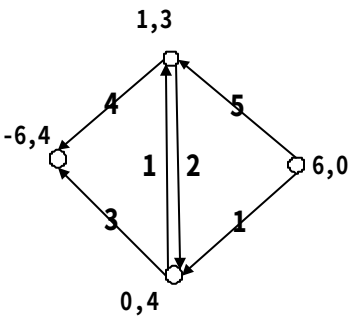
(e)


1,1
4
5
-7,4
1
2
6,0
3
1
0,2

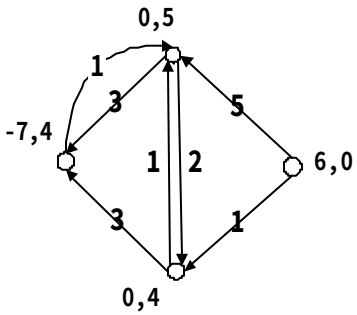
(f)


0,3
4
5
-7,4
2
1
6,0
3
1
1,2

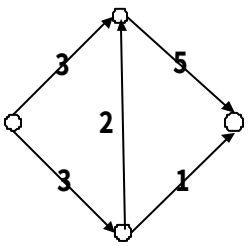
(g)


1,3
4
5
-6,4
1
2
6,0
3
1
0,4

(h)


0,5
1
3
5
-7,4
1
2
6,0
3
1
0,4

(i)


3
5
2
3
1

(j)

图 4.15

初始网络如（a）所示，经过预流推进算法（b）－（i），得到最大流（j）。

b.最高标号预流推进算法[7]

一般的预流推进算法的瓶颈在于非饱和推进。最高标号预流推进算法的思想是从具有最大距离标号的活跃节点开始预流推进，使得距离标号较小的活跃节点累积尽可能多的来自距离标号较大的活跃节点的流量，然后对累积的盈余进行推进，这样可能会减少非饱和推进的次数。

下面是几种预流推进算法实验测试的比较，其中 Rcount 表示重新标号的次数，Scount 表示饱和推进的次数，Ucount 为非标号推进的次数

	队列（秒）	最高标号（秒）	当前流量优先（秒）	栈（秒）	广度优先（秒）
M10	<0.01	<0.01	<0.01	<0.01	<0.01
Rcount	74	210	189	164	151
Scount	219	244	240	201	260
Ucount	313	410	524	1067	435
M30	0.01	0.01	<0.01	0.01	0.01
Rcount	1504	6443	2938	4370	4393
Scount	2107	2576	2116	2068	2958
Ucount	6622	12116	10859	31567	11410
M50	0.03	0.15	0.06	0.17	0.03
Rcount	2866	53146	9024	22863	6263
Scount	4756	8860	4916	5198	5948
Ucount	16148	81798	34024	117985	21256

第五章 最小费用流问题

5.1 最小费用流问题的数学描述

为了描述最小费用流问题，我们首先介绍几个基本的概念。

容量－费用网络：在以 V 为节点集，A 为弧集的有向图 G=(V,A)上，定义弧上的权函数 L、C 和 U，分别表示弧上的单位流量的成本或费用、容量下界和容量上界，定义顶点上的权函数 D，表示顶点的供需量，此时所构成的网络称为容量-费用网络，可以记为 N=（V,A,C,L,U,D)。容量-费用网络也是一种流网络，所以以后我们仍然把容量-费用网络称为流网络或直接称为网络。在上一节中关于网络流的各种概念仍然可以引入到流量-费用网络中。

最小费用流问题就是在这样的网络中，寻找流的容量一定时，总费用最小的可行流。

5.2 关于最小费用流问题的算法

5.2.1 消圈算法

本节考虑传统的单源单汇网络的最小费用流问题，并用 N=（s,t,V,A,C,U）表示以 s 为起点，t 为终点的流网络，其中 V 为节点集合，A 为弧集合，C 为单位费用，U 为流量上界。最小费用流问题就是在网络 N=（s,t,V,A,C,U）中计算流值为 v 的最小费用流 x，或者当不给定流值时，计算流值最大的最小费用流 x。

对网络 N=（s，t，V，A，C，U）中给定的 s-t 可行流 x，网络 N 关于流 x 的残量网络 N（x）=（s，t，V，A（x），C（x），U（x)），其中 A（x），C（x），U（x）定义如下：

$$A(x)=\{(i,j)\mid (i,j)\in A,\ x_{ij}<u_{ij}\}\cup\{(i,j)\mid (j,i)\in A,\ x_{ji}>0\},$$

$$c_{ij}(x)=\begin{cases} c_{ij}, & (i,j)\in A,\ x_{ij}<u_{ij},\\ -c_{ij}, & (j,i)\in A,\ x_{ji}>0。\end{cases}$$

$$u_{ij}(x)=\begin{cases} u_{ij}-x_{ij}, & (i,j)\in A,\ x_{ij}<u_{ij},\\ x_{ij}, & (j,i)\in A,\ x_{ji}>0。\end{cases}$$

其中 u_{ij}（x）称为弧（i，j）上的残留容量。

于是，对于 N（x）中的任何一个有向圈 W，它一定对应于原网络 N 中的一个增广圈，即可以通过沿 W 对当前流 x 进行增广，获得流值相等的 s-t 可行流 y。定义 W 的费用为 C（W）=Σc_{ij}（x），　（i，j）∈W，则当增广的流量为δ时，c（y）=c（x）+δC（W）。由此可见，只要 N 中存在费用为负数的增广圈 W，则可以通过沿 W 对当前流 x 进行增广，获得流值想等但费用更小的 s-t 可行流 y。

设 x 为可行流，则 x 为最小费用流的充要条件是不存在负费用增广圈。

根据这一结论，可以设计如下的消圈算法

消圈算法（在网络 N=（s，t，V，A，C，U）中计算流值为 v 的最小费用流 x）

0：在网络 N=（s，t，V，A，C，U）中计算流值为 v 的可行流 x。

1：在残量网络 N（x）中判别负圈。若无负圈，则已经找到了最小费用流，结束；否则转 2；

2：沿找到的负圈增广流量，转 1。

复杂度分析：步骤 1 中的判别负圈可以采用最短路算法，其复杂度为O（nm）。由于可行流的费用不可能超过 mCU，而每次消去一个负圈至少使得费用下降一个单位，因此消去负圈的步骤 1 和步骤 2 最多执行O（mCU）次。由此可知，该算法的复杂度为O（nm^2CU）。

下面以一个简单的例子来说明消圈算法，用消圈算法计算图 5.1（a）中流值为 6 的最小费用流。首先计算流值为 6 的一个可行流 x_1，如图（b），它的残量网络如图(c)，找到负费用增广圈 2-3-4-2（费用为 2+1-5=-2），增广一个单位的流量，得 x_2 如图（d），它的残量网络 N（x_2）如图（e），在（e）中找到负费用增广圈 2-1-3-4-2（费用为 -1+2+1-5=-3），增广一个单位的流量，得 x_3 如图(f)，它的残量网络中没有负费用增广圈，因此 x_3 是最小费用流。

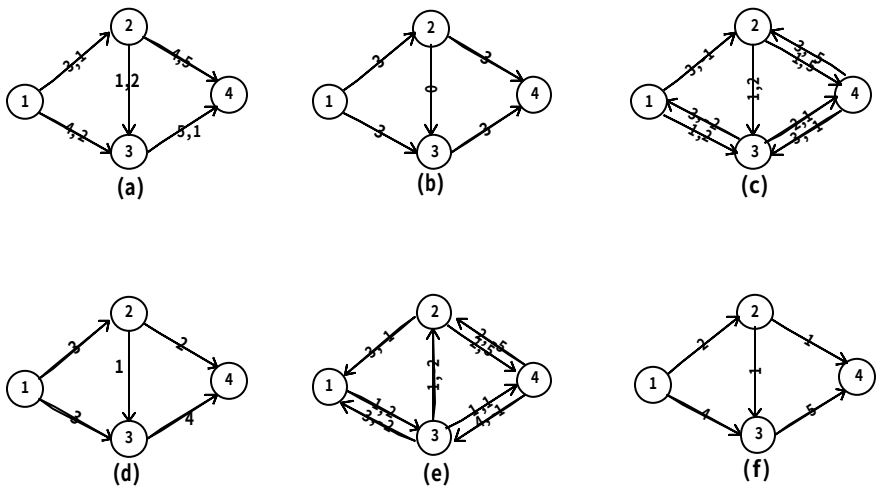


图 5.1 消圈算法的例子

下面是消圈算法运行一些实际问题时花费的时间（秒）。运行环境是 P4 1GHz，256M RAM。

节点数目	200	300	1000
弧的数目	1300	6300	20000
运行时间	127	1463	137544

从上面的实验结果可以看出，消圈算法的实现效率比较低，不适用于解决大规模的实际问题，这主要是因为消圈算法要调用 Bellman-Ford 算法来查找负费用圈，由此可见，要提高消圈算法的效率，应该从优化 Bellman-Ford 算法入手。

5.2.2 最小费用路算法

为了得到流值为 v 的最小费用流 x，我们首先在网络 N=（s，t，V，A，C，U）中计算流值为 v' 且费用最小的 s-t 可行流（$v'<v$），然后对它沿增广路增广以增加流值。

设 x 为流值为 v 的最小费用流，P 为关于 x 的从 s 到 t 的最小费用增广路，且沿 P 所能增广的流量为 δ，则增广后得到流值为 v+δ 的最小费用流。

根据以上讨论，可以构造如下最小费用路算法。

最小费用路算法（在网络 N=（s，t，V，A，C，U）中计算流值为 v 的最小费用流 x）

0：取 x 为任一 s-t 可行流，且在同一流值的流中它是费用最小的流。

1：若 x 的流值达到 v，计算结束；否则在残量网络 N（x）中判别最小费用路。若无这样的路，则流值不可达到 v，计算结束；否则继续 2。

2：沿该最小费用增广路增广流量（增广后的流值不超过 v），转 1。

复杂度分析：算法的主要工作量在于连续寻找最小费用路并增广。由于每次增广最多使得流值增加一个单位，因此步骤 1 和步骤 2 最多执行 O（v）次。由此可见，该算法的复杂度为 O（v）乘上最短路算法的复杂度。记求解非负弧长网络的最短路算法的复杂度为 S（n，m，C），则在残量网络中应用最短路算法的复杂度为 S（n，m，n，C），因为残量网络中弧上的费用上界为 nC。又因为最大流流值不超过 nU，所以本算法复杂度为 O（nU S（n，m，n，C））。

下面使用最小费用路算法求解如图 5.2（a）中流值为 6 的最小费用流，首先从流值为 0 的一个可行流 x_0 开始，此时的残量网络就是原网络，从 1 到 4 的最小费用路为 1-3-4（费用为 2+1=3），沿此路径增广 4 个单位流量得 x_1，如图 5.2(b)，它的残量网络 $N(x_1)$ 如图（c）；找到最小增广路 1-2-3-4（费用为 1+2+1=4），增广 1 个单位得流量，得 x_2 如图（d），它的残量网络 $N(x_2)$ 如图（e）；找到最小增广路 1-2-4（费用为 1+5=6），增广 1 个单位的流量，得 x_3 如图 5.2(f)，此时，已经得到流值为 6 的可行流，因此 x_3 是流值为 6 的最小费用流。

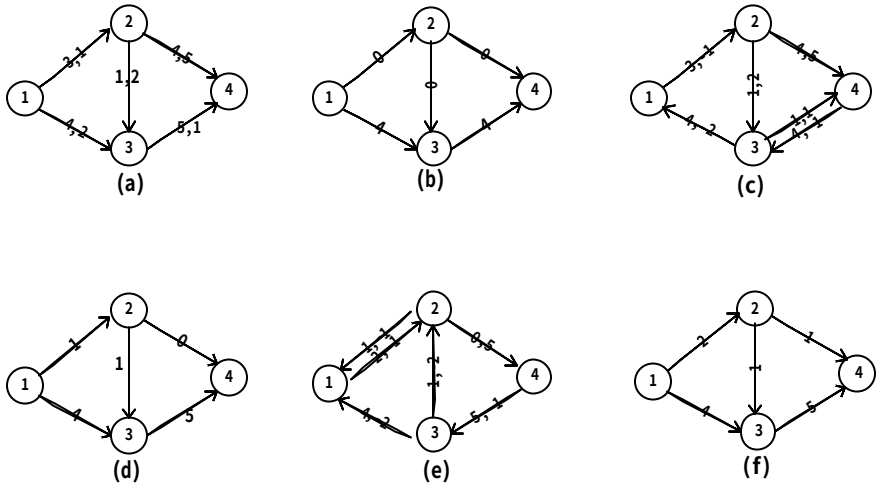


图 5.2 最短路算法的例子

从最小费用路算法的思想看来，它是一种贪婪算法，当每次增广流量都选择费用最小的路径时，有可能在最小费用路网络中的流量达不到最大值。

下面是最小费用路算法运行一些实际问题时花费的时间（秒）。运行环境是 P4 1GHz，256M RAM。

节点数目	200	300	1000	1000	5000
弧的数目	1300	6300	20000	50000	60000
运行时间	1	2	74	88	195

5.2.3 原始-对偶算法

设网络的关联矩阵记为 B，流向量记为 x，则流量守恒条件可以写成 Bx=d。容量约束条件可以写成$-x_{ij}>-u_{ij}$。对这两个约束分别引入对偶变量$?$和 z。根据线性规划对偶理论，如果 x 为原问题的可行解，$?$和 z 为对偶问题的可行解，则它们分别是所对应的问题的最优解的充要条件是它们满足以下的互补松弛条件：

x_{ij}（$?_i-?_j-z_{ij}-c_{ij}$）=0，（i，j）∈A，

z_{ij}（$x_{ij}-u_{ij}$）=0，（i，j）∈A，

在这里，只需要令$z_{ij}=\max\{?_i-?_j-c_{ij}, 0\}$，（i，j）∈A，上面的条件可以等价地写成

当$?_i-?_j<c_{ij}$时，$x_{ij}=0$；（5.1）

当$?_i-?_j>c_{ij}$时，$x_{ij}=u_{ij}$；（5.2）

当 $0< x_{ij}< u_{ij}$时，$?_i-?_j=c_{ij}$。（5.3）

设 x 为最小费用流问题的可行解，则 x 为最小费用流的重要条件是：存在节点的势$?$，满足条件（5.1）--（5.3）。

与最小费用路算法的思路类似，原始-对偶算法也是从任意一个 s-t 可行流，但流值不一定达到 v 的流开始迭代，直到流值达到 v 为止。迭代过程包括两个方面：一是对弧上流的增广，一是对节点上势的修改，并且在迭代过程中始终保持最优性条件（3.1）-（3.3）成立。

通常将残量网络 N（x）中满足$?_i-?_j=c_{ij}$ 的弧组成的子网络称为允许网络，原始-对偶算法就是在允许网络 N^0（x）中寻找增广路并进行增广。当 N^0（x）中找不到增广路且流值小于 v 时，必须进行下一个过程，修改节点上的势。

原始-对偶算法（在网络 N=（s，t，V，A，C，U）中计算流值为 v 的最小费用流 x）

0：取 x 为任一可行流（如 x=0），令初始势?=0.

1：若 x 的流值达到 v，计算结束；否则在残量网络 N（x）中，以 $c'_{ij}=c_{ij}-?_i+?_j$ 为（i，j）弧的弧长，计算从节点 s 到所有节点 I 的最短路路长 d（i），并令$?_i=?_i-$ d（i），继续 2。

2：在允许网络 N^0（x）中判别从 s 到 t 的最大流。若该最大流流值为 0，则原网络中流的流值不可达到 v，计算结束；否则沿该最大流去定的增广路增广流量（增广后的流值不超过 v），转 1。

为了说明算法是正确的，我们首先给出最优性条件（5.1）-（5.3）的一个等价条件：

最优性条件（5.1）-（5.3）等价于对于 N（x）中任意的（i，j）弧，$c'_{ij}\geq 0$。

下面的讨论说明算法终止时一定求到了最小费用流。

在算法运行过程中，最优性条件（5.1）-（5.3）成立。

最后，我们还需要证明，如果将$?_i$ 修改为$?'=?_i-$d（i），则将会有新的弧增加到 N^0（x）中。这可以从下面得到：

对于 N（x）中一条弧（i，j），如果它位于从起点 s 到某一节点的最短路上，则 $c''_{ij}=0$。

算法复杂度分析：从算法的过程可以看到，每次循环迭代修改弧上的流值和节点上的势各一次。由于流值不可能超过 nU，且任何节点上的势不可能低于-nC，因此总的迭代次数不会超过 min{nU，nC}。记求解非负弧长网络的最短路算法的复杂度为 S（n，m，C），最大流算法的复杂度为 M（n，m，U），则本算法复杂度为O（min{nU,nC}[S（n，m，nC）+ M（n，m，nU）]）。

为了让大家更容易理解这个算法的计算过程，下面我们用原始-对偶算法计算一个例子。计算图 5.3（a）网络中的最小费用流，图中弧上的前一个数字表示弧上的容量，后一个数字表示弧上的单位费用；节点上的数字表示节点的供需量。

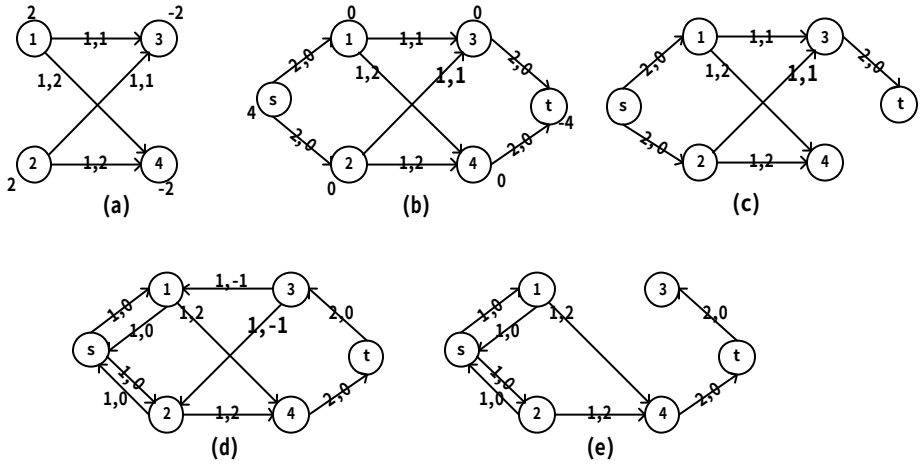
我们下面根据算法的具体步骤进行计算：首先引进两个新节点（源 s 和汇 t），可以把问题转化为计算单源单汇的流值为 4 的最小费用流问题。即网络 N 如图（b）。

令初始流 x=0，初始势?=0，此时的残量网络 N(x)=N。以 $c_{ij}'=c_{ij}-?_i+?_j=c_{ij}$ 为弧长，计算从 s 到各节点的最短路路长为：d(s)=d(1)=d(2)=0，d(3)=1，d(4)=2，d(t)=1。写成向量的形式，即 d=(0,0,0,1,2,1)。修改?得到?=(0,0,0,-1,-2,-1),此时允许网络 N^0(x)如图 5.3(c)。

在此时网络 $N^0(x)$中，计算从 s 到 t 的最大流流值为 2（沿 s-1-3-t 和 s-2-3-t）各发送一个单位的流量）。此时残量网络 N(x)如图 5.3(d)。

以 $c_{ij}^{?}=c_{ij}-?_i+?_j$ 为弧长，计算从 s 到各节点的最短路路长为 d=(0,0,0,1,0,1)。修改?得到?=(0,0,0,-2,-2,-2)，此时允许网络 $N^0(x)$如图 5.3(e)。

在此时网络 $N^0(x)$中，计算从 s 到 t 的最大流流值为 2（沿 s-1-4-t 和 s-2-4-t 各发送一个单位的流量）。此时各节点流量达到平衡，即已经得到最小费用流为 $x_{13}=x_{14}=x_{23}=x_{24}=1$。



下面是原始-对偶算法运行一些实际问题时花费的时间（秒）。运行环境是 P4 1GHz，256M RAM。

节点数目	200	300	1000	1000	5000
弧的数目	1300	6300	20000	50000	60000
运行时间	1	5	132	271	1042

5.2.4 几种算法的比较

从上面的实验结果可以看出，我们在解决较大规模的实际问题时，最短路算法和原始-对偶算法是可行的，其中最短路算法效率最高，虽然它可能会浪费一些流量，但是浪费的比例一般都很小，从我们多次运行的例子看来一般不会超过 10%，是可以接受的。由于最小费用流算法的计算过程中要调用最大流算法，最短路算法等其它比较基础的算法，所以我们在实现最小费用流算法的时候要注意选择效率高的最大流算法和最短路算法，这样才不至于因为其他算法的效率差异对我们的最小费用流算法的实现效率产生太大的影响。

5．3 实验网络生成工具的介绍

本文中介绍的系统能够根据用户的需要来随机生成不同规模的网络，用于作为本系统的测试输入，由于我们的生成工具采用的是目前通用的格式，所以也可已经用于其它求解器的测试输入。

5.3.1 网络生成部分的输入参数

系统中的测试网络的生成器的输入主要包括以下参数：网络中的节点总数；网络中源节点总数；网络中汇节点总数；网络中弧的总数；弧的最大费用值；弧的最小费用值；总的运输量；转运点中源节点的总数；转运点中汇节点总数；费用为最大费用值的弧所占的百分比；容量有所限制的弧所占的百分比；限制弧的容量的下限值；限制弧的容量的上限值。

网络生成部分根据上面的这些输入数据的不同，可以生成三大类问题的网络：运输问题、指派问题和最大流问题。具体的情况如下，当源节点总数与汇节点总数之和等于节点总数，并且转运点数目等于零的情况下，生成运输问题；当源节点总数等于汇节点总数，并且总的运输量等于源节点的总数时，生成指派问题；当弧的最小费用等于弧的最大费用，并且不属于指派问题时，生成最大流问题。

5.3.2 网络生成部分的输出形式

目前系统生成的实验网络都是以文件的形式保存，系统中的算法运行部分按照规定的格式读取文件中的数据。

文件格式的定义是这样的，每一行的开头都有一个字符来表示该行代表的意义，网络中的节点用整数 1 到 n 表示，网络中的弧用该弧的头节点和尾节点构成的整数对表示。具体的规定如下：

c <文本>，字符 c 表示该行是注释行；

p 问题类型，<网络中节点总数><网络中弧的总数>，字符 p 表示这一行用于问题的总体描述，其中“问题类型”分别用 asn，max，min 表示指派问题，最大流问题，最小费用流问题；

n <节点号><该节点的流值>，字符 n 表示这一行用于描述节点，如果该节点的流值大于 0，表示它是源节点；反之，是汇节点；

a <该弧的尾节点><头节点><容量下界><容量限制><该弧的费用>，字符 a 表示这一行用于描述一条弧，后面是这条弧的相关参数，如果容量限制小于 0，说明这条弧不可用。

下面是一个生成的随机网络的例子，前面以字符 c 开头的注视部分说明了生成这个随机网络时输入的相应参数。

```
c    Random seed:            13502460
```

```
c   Number of nodes:              200
c   Source nodes:                 100
c   Sink nodes:                   100
c   Number of arcs:              1300
c   Minimum arc cost:               1
c   Maximum arc cost:           10000
c   Total supply:              100000
c   Transshipment -
c     Sources:                      0
c     Sinks:                        0
c   Skeleton arcs -
c     With max cost:                0%
c     Capacitated:                  0%
c   Minimum arc capacity:           0
c   Maximum arc capacity:           0
p min 200 1308
n 1 286
n 2 671
.....
n 199 -131
n 200 -124
a 1 102 0 100000 8550
a 1 171 0 100000 7415
.....
```

第六章 系统的设计与实现

6．1 总体设计思想

网络优化算法的实现与比较系统是将图论中网络优化部分的理论应用于实际的一个计算机软件系统。它解决的主要问题是与图论相关的几类问题，可以解决许多模型化之后的实际问题，比如运输问题、连通问题、公路连接问题等等。系统的设计采取由简单到复杂的设计思路，并充分考虑到系统的可移植性，提供了可供外部调用的算法接口，可以作为一个算法库集成到其它的应用系统中。另外，系统还集成了一个网络样本的生成器[8]，可以根据需要来生成各种规模的实验样例，来作为系统测试和评估的输入数据。

该系统采用 Microsoft Visual C++为开发平台，采用多文档界面结构。系统提供了比较友好的人机交互接口。需要求解的问题以通用的标准格式存放在文件中，系统提取数据以后，运行相应的求解算法，然后把运行结果存放到文件中，同时在程序运行窗口显示给用户。

从系统结构的角度看，系统共由四大模块组成，分别是问题生成模块、最短路问题求解模块，最大流问题求解模块和最小费用流问题求解模块。

问题生成模块能够根据用户的需要把抽象的问题以标准的表示方式表示出来，作为运行相应算法的输入。根据输入的参数不同，问题生成器能够分别生成运输问题、指派问题或者最大流问题。

最短路问题求解模块提供了求解最短路问题的两种典型算法，即 Dijkstra 算法和 Bellman-Ford 算法，通过比较，可以看出这两种算法各有优缺点，Dijkstra 算法运行速度比较快，但是适用范围不是很广；Bellman-Ford 算法虽然运行速度相对较慢，但是能够解决所有的最短路问题。这个模块中实现的算法也是为后面求解最大流问题和最小费用问题做铺垫，因为在后面较为复杂的算法中，可能需要调用最短路算法，这时候，为了尽量减小最短路算法效率对相应算法效率的影响，我们应该在可用的算法中尽量选择效率好的算法。

最大流问题求解模块提供了求解最大流问题的多种算法，通过我们提供的对这些算法的比较，用户可以选择比较适合的算法来求解，需要指出的是，即使是同一种算法，由于数据在计算机中的表示方式和存储方式不同，算法的执行效率也有很大的差别。

最小费用流求解模块提供了求解最小费用流问题的三种典型算法。前两种算法的思想比较容易理解，实现起来也相对容易一些，但是在执行效率上不如后面的对偶算法好。以上三类问题的输出目前都是以标准的格式存放在文件中，同时在程序运行窗口显示给用户，也可以根据需要保存到数据库中去。

6．2 系统主要功能

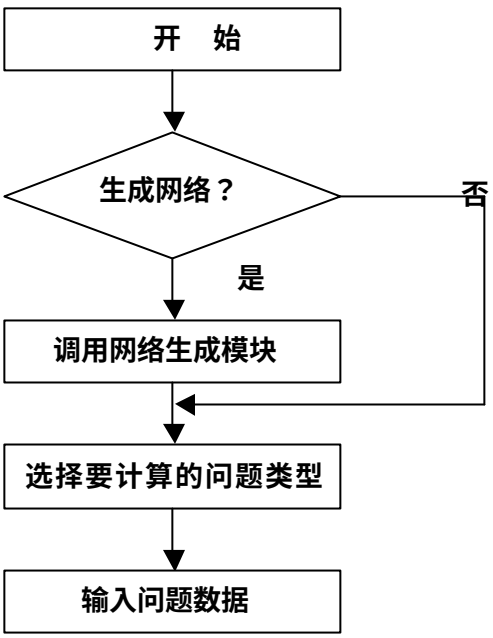
该系统的主要功能是解决与网络优化相关的实际问题，系统分为求解最短路问题、求解最大流问题、求解最小费用问题和实验网络生成等四个主要功能。

最短路问题的求解部分可以解决与最短路问题相关的一些实际应用问题，比如在交通网络中寻找两点之间的最短路径，可以在单位距离运输费用相同的情况下，选择一条比较好的路线；最大流问题的求解部分能够在给定的网络中，使用不同的算法求出当前网络种可能达到的最大流，同时可以得到网络的最小切，这在解决一些实际问题时是十分有效的。与最大流问题相关的典型问题包括水库的供水区域划分问题，公路网上收费站的建设规划问题等等；最小费用路问题的求解部分提供了解决最小费用路相关问题的解决途径，可以解决运输问题等。

上面提到的解决三类问题的相应算法的实现代码都为外部调用提供了简洁明了的接口，用户也可以把这部分代码作为一个算法库集成到其它的实用系统中去。

网络生成部分能够根据需要，由用户自定义生成特定的网络，用来表示指派问题、最大流问题和最小费用流问题，以此作为验证算法的实验输入，为了保证实验的随机性和可信性，我们在生成网络的时候应用一个随机数作为输入，网络中的弧和弧上的参数值都是随机产生的，生成的网络存放以文件的形式保存起来，同时显示给用户，用户可以根据实际需要对网络进行适当的调整

6．3 系统主要流程



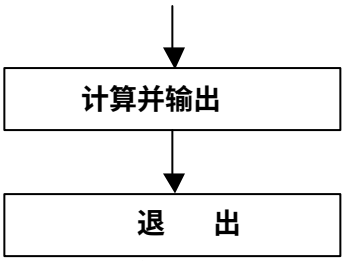


图 6.1 系统的主体流程

6．4 系统主要界面

1．运行系统的主窗口

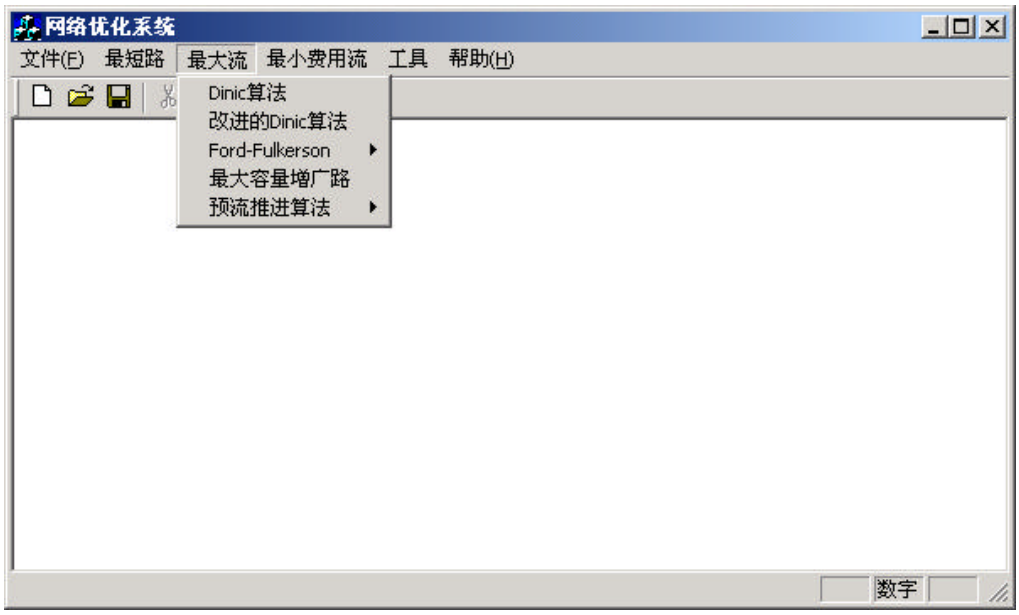


图 6.2 系统主界面

2．网络生成部分的界面

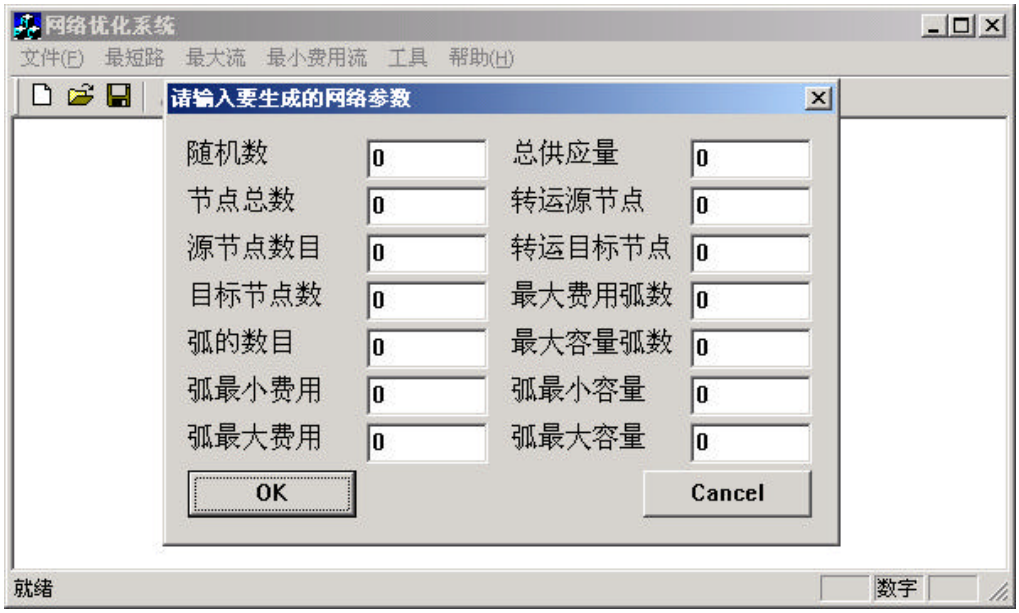

网络优化系统
文件(F) 最短路 最大流 最小费用流 工具 帮助(H)
请输入要生成的网络参数
随机数
0
总供应量
0
节点总数
0
转运源节点
0
源节点数目
0
转运目标节点
0
目标节点数
0
最大费用弧数
0
弧的数目
0
最大容量弧数
0
弧最小费用
0
弧最小容量
0
弧最大费用
0
弧最大容量
0
OK
Cancel
就绪
数字

图 6.3 **网络生成界面**

第七章 结束语

7．1 系统评价

虽然本文所讨论的网络优化的实现系统的目的是追求最佳的网络优化方案，使运输网络发挥出最高的效率。然而这只是一种理想化的情况，实际应用中存在许多特殊情况，可能达不到最优解。但是，该系统能够得到较好的网络优化解决方案，尽量地提高网络利用效率，从而达到降低运输费用等目的。

一般说来，复杂的算法或者程序的复杂度证明是非常困难的。算法的复杂度分为时间复杂度和空间复杂度，衡量算法的好坏要从这两方面综合考虑。单源最短路算法的空间复杂性为 $O(n^2)$，最大流算法的空间复杂性为 $O(n+e)$，时间复杂性各有不同。虽然例子不能作为衡量一个算法好坏的最终标准，但作为比较的目的，使用若干样例说明或测试一个算法在实际应用中的好坏，也是一种可取的方式。程序的具体实现方法对算法的执行效率也是有非常大的影响的，而算法的最终应用总是由程序来实现。

本实验室开发的“物流决策支持系统”，其中制定物资运输方案和制定物资调拨方案的核心就是最短路算法、最大流算法和最小费用流算法。这对于企业节省运输成本、提高运输效率起着关键的作用，在实际优化运输网络中有重要的实际意义的。系统采用目前被广泛采用的 DIMACS 规定的标准输入方式。也可以根据需要扩展为从数据库中提取数据。系统提供了灵活的外部调用接口，可以作为一个函数库集成到其它的实现系统中。系统的算法库也可以扩充，通过加入新的算法来增强功能，最终使系统成为能解决网络优化中更多更复杂的问题。

7．2 存在的问题

由于实际的需要以及问题的复杂性，网络优化算法的实现系统还存在着许多问题需要进一步解决。

从理论角度上讲，有关网络优化算法的研究一直在进行着，不断有新的改进算法提出，尤其是针对一些特定问题，采用特定的算法执行效率可能提高很大，而目前我们的系统中实现的算法都是一些主流的经典算法。因此，要不断地充实我们的算法库，设法提高问题的求解速度。

从实际的应用来看，要应用我们的系统解决实际生活中各种各样的问题，还需要对问题进行模型化，这个模型化的过程我们做得还不是很好，需要进一步的改进。

虽然网络优化的理论算法与实际应用之间还存在着一定的差异，但是网络优化的

实际需求是非常迫切的，尤其是因为近年来有关物流配送方面的兴起，即使是比较简单的网络优化策略，也会带来巨大的经济效益。

7．3 进一步的工作

该系统目前还处于实验室开发阶段，还需要实际数据进一步的检验；目前实现的算法种类也有限，还需要不断扩充，以使该系统能够解决更多的特定问题，使系统不断得到完善；另外，具体问题的模型化模块也需要不断扩充和完善，以使网络优化的执行效率更高，结果更精确。

另外，该系统目前主要是针对运输问题开发的，也可以将该系统稍加改进而应用于其它的优化问题，如资源分配问题等等。

参 考 文 献

[1] A new approcach to the maximum-flow problem, Goldberg , Andrew V. Tarjan, Robert E.Jouranal of the ACM.Vol. 35, No.4（Oct. 1988）,.921-940

[2] **刘家状，徐源。网络最优化。高等教育出版社，**1991

[3] **谢金星，邢文训。网络优化。清华大学出版社，**2000，160-168

[4] **田丰，马仲蕃。图与网络流理论。科学出版社，**1995

[5] **谢政，李建平。网络算法与复杂性理论。国防科技大学出版社，**1995

[6] **卢开澄。计算机算法导引：设计与分析。清华大学出版社，**1996

[7] *Dimacs Implementation Challenge Workshop, Algorithms for Network and Matching,* DIMACS Technical Report 92-4.

[8] E.A. Dinic *Algorithm for solution of a problem of maximum flow in networks with power estimation,* Soviet Math. Dokl. 11 (1980).

[9] Karzonov, A.Determining the maximal flow in a network by the method of preflows. *Soviet Math. Doklady* 15 (1974), 434-437.

[10] A.V. Goldberg/R.E. Tarjan: *A New Approach to the Maximum Flow Problem,* Proc. 18th ACM Symp. on Theory of Computing, 1986.

[11]Klingman, D., A. Napier, and J. Stutz, "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems", Management Science 20, 5, 814-821 (1974)

[12] Ahuja R K，Magnanti T L and Orlin J B。Network Flows:Theory, Algorithms, and Applications。Englewood Cliffs,New Jersey：Prentice Hall 1993

[13] Bazaraa M S, Jarvis J J and Sherali H D。Linear Programming and Network Flows。2nd ed。New York：John Wiley&Sons, 1990

[14] Evans J R，Minieka E。Optimization Algorithms for Networks and Graphs。2nd ed。New York：Marcel Dekker Inc.，1992

[15] Papadimitriou C H，Steiglitz。Combinatorial Optimization：Algorithms and Complexity。Englewood Cliffs，New Jersey：Prentice Hall，1982

摘　　要

我们生活在一个网络社会中，从某种意义上说，现代社会就是一个复杂的网络系统。网络优化就是研究如何有效地计划、管理和控制这个网络系统，使之发挥最大的社会和经济效益，它是运筹学中的一个经典和重要的分支。

目前，物流配送越来越受到人们的重视，在美国，全部生产过程只有 5%的时间用于加工制造，余下的 95%时间都是储存和运输时间，因此，提高运输效率可以极大的缩减商品的流通费用，从而使企业获得巨大的经济效益。在我国，物流配送行业正在兴起，许多城市都在努力成为地区性的物流中心，这就需要相应的计算机软件管理系统，但是目前我国的软件市场上还没有得到广泛认可的货物运输网络优化系统。

本文讨论了“网络优化算法的实现与比较系统”的设计要点与实现过程，以及围绕图论中的几类典型问题的相关理论研究工作。本文从建立实用的网络优化系统出发，介绍了开发网络优化系统的背景和目的，并回顾了网络优化问题研究的历史和现阶段的研究状况，阐明了开发网络优化系统的必要性及其广泛的适用性。

本文对网络优化中的几类问题进行了深入的分析，分别介绍了关于最短路问题、最大流问题和最小费用流问题的数学描述和几种典型的算法，这三类问题都是图论中的典型问题，我们着重介绍了这些问题的数学描述和各类算法的基本思想和算法的详细描述，由于篇幅的限制，我们没有对算法的正确性和复杂性做出证明，只是给出了结论。由于处理这些问题的算法非常多，而且新算法还在不断涌现，我们只能有选择的介绍其中部分算法，我们既介绍一些比较经典的算法，也介绍了一些比较新颖、实用的算法。

本文在实现算法的同时，还给出了各种算法的运行效率的比较数据，为了增强我们的实验数据的可信性，我们在选择实验样例时特别注意了样例的随机性和代表性，还考虑到了样例之间的可比性，这对用户使用我们的系统求解网络优化问题时如何选择合适的算法有一定的指导作用。在系统的运行测试过程中，在实验条件允许的范围内，我们进行了多个样例的运行测试，尤其是大规模问题的测试工作，通过实验结果，我们认为本系统在解决实际的应用问题时，运行的时间代价是可以接受的，效率是令人满意的。在实现系统和测试系统的过程中，我们注意到，算法的执行效率会受到很多外部因素的影响（数据的表示方式，底层函数的执行效率等），因此，要客观的评价一个算法的优劣，要尽量把这些外部因素的影响降到最低限度。另外，关于某一类问题的通用算法虽然适用的范围很广，但是一般说来，效率都不是很突出，针对特定的问题选择特殊的算法，执行效率可能会有很大的提高。

在已有的经典算法的基础上，本文给出了对几种算法的改进算法，包括对 Dinic 算法的改进算法，通过改进网络的表示方式（给每条弧增加一个标志位），减少了构造新网络和拷贝网络流的时间，使算法的执行效率有了很大的提高；在对 Bellman-Ford

算法修改后，用于查找网络中的负费用圈，应用于解决最小费用流问题的消圈算法中；另外，对预流推进算法中的几种选择活跃节点顺序的策略，我们也进行了比较和讨论。

本文还介绍了网络优化算法的实现与比较系统的总体设计思想，系统的框架结构系统的主要功能和系统的主要界面等。该系统立足于基本的网络优化算法，采用由简入繁的设计思想，充分考虑到系统中算法的可扩充性，使系统能随着网络优化算法的不断丰富而日臻完善。另外，本文总结了建立该系统的过程中遇到的困难以及进一步要做的工作。

另外，本文还介绍了系统中的样例网络生成器，这个生成器能够根据用户的要求来生成不同规模的网络，然后按照 DIMACS（Center for Discrete Mathematics and Theoretical Computer Science）规定的格式标准以文件的形式保存，这些例子都是随机生成的，可以用于本系统的算法实现测试，或者提供给其它系统作为测试样本。

尽管该系统目前实现的算法数量有限，能够解决的问题种类不是很多，但是系统的执行效率是能够满足解决一般的实际问题的，这就使该系统有了足够的应用价值，随着更多的、执行效率更高的网络优化算法不断补充，该系统也将不断完善，从而使应用更广泛、更实用。

Abstract

We live in a network world today, for some meaning, modern society is a complex network system. Network optimization is a subject about how to scheme, manage and control this system more efficient, and then we can get the maximum social benefit and economical benefit. Network optimization is a classical and important branch of operational research.

Material flowage get more and more attention now, there is only 5% total produce time is used to manufacture, the other 95% is used to store and transport those ware. So we can hold down the current cost by increasing transport efficiency, and the enterprise can get greatness economical benefit. Material flowage is spring up in our country, many cities are trying to be regional center of material flowage, all these need responding computer manage systems, but there is not any widely approbated software system about carrying network optimization.

This paper discussed the design outline and implement process of the Implementation and Comparison of Network Optimization Algorithms System, and some academic research about some typical problems of chart theory, from the point of the establishment of applied network optimization system, we introduced the background and the purpose of the development of network optimization system, looked back the history of network optimization research, illuminated the necessary of network optimization system and its widely applications.

This paper analyzed several kinds of problem about network optimization, and introduced the mathematical description and some typical algorithms about the shortest path problem, the maximum flow problem and the minimum cost flow problem. All these problems are typical problems of chart theory, we emphasize introduced the mathematical description and the algorithms, because of the restriction of the length, we didn't prove the correction and the complexity of these algorithms, but we introduced correlative conclusions. Because there are lots of algorithms about these problems, we introduced part of these algorithms selectivity, we introduced some typical algorithms and some novel algorithms.

On the base of these classical algorithms, this paper provides some improved algorithms. Including the improved algorithms about Dinic algorithms, we get distinct improvement by mending the express method of graph (added a

superfluity flag parameter), we decreased construct graph and duplicate graph times; we amended the Bellman-Ford algorithm, and used it to find the minus cost circles in graph, applied it in the erase circle algorithms about the minimum cost problem; we compared and discussed these strategies about the selection of active nodes in the pre-flow push algorithms.

This paper presented the compare data of the execute efficiency of these algorithms. To make the test data more reality, we noticed the randomicity and the representation of the test examples, these data can help user select right algorithms when they use our optimization system. During the test process, we did lot execute test, especially big scale questions, according these test data, we consider that the execute efficiency is acceptable. During the implement procedure and the test procedure of this system, we noticed that the efficient of algorithms be infected by many exterior conditions (denote method of test data, lower functions, etc), so to opinion an algorithm impersonal, we should do our best to decrease these infections. Otherwise, universal algorithms adapt to common questions, but its efficiency is not very good, select special algorithms to resolve specifically problems, we can get higher efficiency. This paper also summarized the difficulties when we construct this system and those work should be do in future.

This paper introduced the example network generator, this generator can generate network of different scales by the control of users, then save the example network as file in the format prescribed by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), all these examples are generated randomly and can be used to test the algorithms implementation in this system or provide to other systems for test.

The paper presents the design idea, the frame and the main functions of the Implementation and Comparison of Network Optimization Algorithms System. On the base of the elementary network optimization algorithms we design the system from simple to complex. And it is considered that the system is extensible. So it will become perfect with other network optimization algorithms added continually.

Though there are only a few algorithms are implemented in this system, and only a few kinds of problem can be solved by this system, the efficiency of this system can satisfy actual problems, which enables the system to have enough applied value. With coming out of more applicable, more comprehensive network optimization algorithms, the system will become better and better, and will be applied widely, and will be more practical.

致　谢

首先向我的老师孙吉贵教授表示衷心的感谢和尊敬，在我读研期间，老师对我的悉心的知道让我受益匪浅，老师严谨认真的治学态度，平易近人的性格，都给我留下了难以磨灭的记忆。

我还要特别感谢在我读研期间给过我很多帮助的白洪涛、张一民、曹晓威、姜迎新和林海同学，他们营造了良好的学习氛围，并给予我无私的帮助和支持。我们共同学习，度过了一段美好的时光，从他们身上我学到了丰富的知识和宝贵的经验。

衷心和感谢我的父母和所有关心我的亲人和朋友。