线性规划入门（An Introduction to Linear Programming）

目录

出品团队

『运筹OR帷幄』公众号作为华人最大的运筹优化|DS|AI社区之一，有着强大的技术背景和丰富的企业资源。公众号以运筹学/优化为基石，积极拓展数据科学/人工智能等相关领域，致力于打造精准&高度垂直的算法和数据社区，聚集领域内优秀学者、企业和优质人群。

2019年1月，我们在公众号发布了线性规划入门电子书的征稿文，反馈影响热烈，最终由以下同学组成作者团队，共同完成了本书的创作。

作者：

李崇楠（第1章、第5.2章）北京交通大学 交通运输规划与管理专业 研究生在读

孙睿晗（第2章） 荷兰阿姆斯特丹大学 人工智能硕士研究生在读

臧永森（第3章） 清华大学 工业工程系 博士在读，戚铭尧老师团队

翁欣（第4章） 清华大学 博士在读

阎泳楠（第5.1章、第6章） 同济大学 交通运输工程 硕士研究生

魏晓阳（第7章） 新加坡国立大学 土木与环境工程系 博士生在读

在编写过程中，编者参阅了许多国内外的优秀运筹学教材，在此对参阅教材的编者表示衷心感谢。

修改意见

由于编者水平有限，书中内容定有许多不足，本电子书项目会长期维护，欢迎各位读者朋友提出宝贵修改意见&参与修改！

向我们提出意见的方式包括：

① Github平台，pull requests, issues （强烈推荐）

② 发送至邮箱operations_r@163.com

第1章 概述

作者：李崇楠，北京交通大学 交通运输规划与管理专业 研究生在读

研究方向：运输组织优化

本章带领读者走进线性规划的世界。首先介绍线性规划的历史，读者将了解到在线性规划领域作出杰出贡献的学者及其成就；接下来的内容为线性规划的基本概念，模型假设和“标准形”，并介绍了将一个般的线性规划模型转化为标准形的技巧；最后给出了若干线性规划案例。

1.1线性规划的历史

1939年，苏联学者Kantorovich为前苏联政府解决优化问题时提出了极值问题，并且提出了解乘数法的新方法，可惜他的工作在当时并未引起足够的重视。事实上，他所提出的问题正是线性规划的雏形。

与此同时，美国的线性规划却获得了飞快的发展。1941年，Hitchcock提出运输问题；1945年，Stigler提出了营养问题；1945年，Koopmans提出了经济问题。而奠定线性规划整套理论方法的，还要说是G.B.Dantzig，他被誉为“线性规划之父”。他在1947年担任美国空军审计官的数学顾问，为找到解决问题的机制化工具，提出了“在一组线性方程或不等式约束下，求某一线性形式极小值问题的数学模型”，这便是“线性规划”（linear programming）这一经典优化模型。而“线性规划”这一名字的由来是在之后1948年，Koopmans和Dantzig在海滩散步时共同想出的。1947年夏天，Dantzig提出了单纯形算法。这个算法在后来被评为20世纪最伟大的算法之一。

尽管单纯形法(Simplex method)作为解决线性规划的有效方法在学术界具有统治地位，但是1971年，Klee和Minty两位学者构造出一个例子，该例子下单纯形法的运作需要访问指数数量级别的顶点，也就是说，在最坏情况下，单纯形法是一个指数时间算法(exponential-time algorithm)。Dantzig在得知这个消息后感叹到他的噩梦到来了，单纯形法并不是在任何情况下都是高效可行的。那么，是否有更加高效的算法，比如多项式时间算法(polynomial-time algorithm)，来解决线性规划问题呢？8年后，即1979年，L.G.Khachiyan发明了椭球算法(ellipsoid method)，这是第一个解决线性规划问题的多项式时间算法。但是，这个算法虽然理论上是多项式时间运行，但是算法被证明是不切实际的，这个算法的杰出贡献是在理论层面告诉世人，线性规划是可以用多项式时间算法来解决的，同时也启发了学者在更加深入的优化领域进行算法开发。1984年，N.Karmarkar发明了内点算法(interior point method)，这是线性规划第一个实际可用的多项式时间算法。

1.2 线性规划模型

1.2.1 基本概念

线性规划是一类经典的优化模型。与一般的优化模型类型，线性规划模型也有目标函数，决策变量和约束条件。那决定这个优化模型是线性规划的因素是什么呢？下面我们将以一个例子展开介绍。

$$\begin{aligned} \max \quad & x_1 - 2x_2 && (1.1) \\ \text{s.t.} \quad & x_1 + x_2 \leq 40 && (1.2) \\ & 2x_1 + x_2 \leq 60 && (1.3) \\ & x_1, x_2 \geq 0 && (1.4) \end{aligned}$$

上述模型为典型的线性规划模型，式(1.1)是目标函数，x_1, x_2 是决策变量，式(1.2)-(1.4)为约束条件，其中式(1.2)和式(1.3)为线性约束，式(1.4)为非负约束。可以发现，对线性规划模型而言，目标函数和约束条件都是线性函数。线性函数可以理解为每一项未知数(变量)的最高次数为1，即不会出现x_1x_2,x_1^2等未知数次数超过1的情况。

1.2.2 模型假设

线性规划作为一类数学模型，含有以下三种假设：

1.比例假设

在线性规划中，目标函数的系数是固定的常数，但是在现实生活中不见得如此，比如考虑目标函数是计算采购蔬菜计划的总花费。那么可能会面对“2元钱1个，3元钱2个”的这种促销，这时线性规划的目标函数就无法表示。

2.非负假设

线性规划的决策变量要求时非负数，但是不要求是非负整数，所以可以取小数。这同样有局限性，依然以采购蔬菜的问题为例，如果需要买土豆，但是土豆只能按整袋来卖的话，那么表示买土豆数量的决策变量就只能取整数了。

3.确定性假设

线性规划中像是目标函数的系数，线性约束中的常数与决策变量的系数，都是已知且保持不变的常数。但是有些参数在实际中是会变化的，通常参数变化的范围是已知的，这就涉及到鲁棒优化(robust optimization)、随机优化(stochastic optimization)等领域的内容了。

1.2.3 线性规划的标准形

在线性规划中，根据目标函数是试图取得最大还是取得最小，能够分成“最大化问题”与“最小化问题”两种问题。此外，线性约束根据决策变量的线性表达式和常数项之间的连接符为“大于等于”“等于”“小于等于”，又可以分为“大于等于约束”“等于约束”“小于等于约束”。因此，线性规划可以有不同的形式，这不利于定理、算法的表示及推导。为解决这一问题，引入线性规划的标准形。

$$
\begin{aligned}
\max \quad & z = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \\
\text{s.t.} \quad & a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n \quad = b_1 \\
& a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n \quad = b_2 \\
& \vdots \\
& a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n = b_m \\
& x_1 \geq 0, x_2 \geq 0, \cdots, x_n \geq 0
\end{aligned}
$$

观察上面的模型可以看到，线性规划的标准形具有如下特点：

（1）目标函数要取得最大值

（2）所有的决策变量都要满足非负约束（nonnegativity constraint）

（3）线性约束均为等式约束（equality constraint）

如果有一个非标准形的线性规划，那么如何等效地转化为标准形呢？这里要强调所谓“等效”的概念，即最优解是不变的，或转化后问题的最优解能够通过一定的方式推出原问题的最优解。

下面我们介绍将一个线性规划模型转换为标准形式的技巧。

1.3 线性规划标准形式的转换技巧

针对如下线性规划问题，我们将使用一些技巧，将它转化为线性规划模型的标准形式。

$$\begin{aligned}\min \quad & 3x_1 - 2x_2 - 4|x_3| \\ \text{s.t.} \quad & -x_1 + 2x_2 \leq -5 \\ & 3x_2 - x_3 \geq 6 \\ & 2x_1 + x_3 = 12 \\ & x_1, x_2 \geq 0\end{aligned}$$

与标准形进行对比，观察到有如下的差别：

（1）目标函数是最小化，而不是最大化。

（2）线性约束中有大于等于约束，有小于等于约束。

（3）决策变量x_3无约束，我们称之为自由变量。

（4）目标函数含有绝对值项，这是一个很难处理的要点，需要一定技巧才能化解。

下面逐一介绍转化为标准形的技巧。

1.目标函数最大化

针对目标函数是最小化的模型，我们可以将原问题的目标函数乘以负一，并最大化这个新的目标函数。在本例中，目标函数可以转化为

$$\max \quad -3x_1 + 2x_2 + 4|x_3|$$

2.消除不等式约束

我们可以看到第一个约束条件是“小于等于约束”，第二个约束条件是“大于等于约束”。在这里我们分别引入松弛变量（slack variable）进行转化。松弛变量均满足非负约束。对“小于等于约束”，我们加上一个松弛变量s_1；对“大于等于约束”，我们减去一个松弛变量 s_2。这样两个不等式约束都转换成了等式约束：

$$-x_1 + 2x_2 + s_1 = -5$$

$$3x_2 - x_3 - s_2 = 6$$

3.消除自由变量

在本例中，变量x_3没有约束。针对这类无约束变量，我们需要引入两个非负变量x_3^+, x_3^-来表示它：$x_3 = x_3^+ - x_3^-$ 但是这个例子的目标函数中，对x_3取绝对值，所以此处对x_3的转换还需要进一步的操作。

4.消除绝对值符号

在转换的时候，我们需要对绝对值符号内的项进行正负号判断，将原本含有绝对值符号的式子拆分成两个不含绝对值符号的式子。考虑到绝对值符号可能出现在目标函数中，也可能出现在约束条件中，接下来我们给出两种示例。

(1)目标函数中有绝对值符号

本例中目标函数值的第三项含有绝对值符号$|x_3|$,其中x_3为自由变量。我们引入两个变量x_3^+, x_3^- ,定义为:

$$x_3^+ = \begin{cases} x_3 & \text{if } x_3 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$x_3^- = \begin{cases} -x_3 & \text{if } x_3 \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

由变量定义可知，x_3^+, x_3^-一定都是非负的，并且$x_3 = x_3^+ - x_3^-, |x_3| = x_3^+ + x_3^-$。 读者可以令$x_3$取某一具体的数值来验证。如当$x_3 = 4$时，
$x_3^+ = 4, x_3^- = 0, x_3 = x_3^+ - x_3^- = 4 - 0 = 4, |x_3| = x_3^+ + x_3^- = 4 + 0 = 4$; 当
$x_3 = -4$时，$x_3^+ = 0, x_3^- = 4, x_3 = x_3^+ - x_3^- = 0 - 4 = -4,$
$|x_3| = x_3^+ + x_3^- = 0 + 4 = 4$。

(2)约束条件中有绝对值符号

如有以下约束$|-x_1 + 2x_2| \leq 7$。可以将其转化为：

$$\begin{cases} -x_1 + 2x_2 \leq 7 & \text{if } -x_1 + 2x_2 \geq 0 \\ x_1 - 2x_2 \leq 7 & \text{if } -x_1 + 2x_2 \leq 0 \end{cases}$$

然后对上述式子添加松弛变量，转换成等式约束。

所以总结起来，我们可以把原线性规划问题转化为如下的标准形式：

$$
\begin{aligned}
\max \quad & -3x_1+2x_2+4x_3^+ +4x_3^- +0s_1+0s_2 \\
\text{s.t.} \quad & -x_1+2x_2+s_1=-5 \\
& 3x_2-x_3^+ +x_3^- -s_2=6 \\
& 2x_1+x_3^+ -x_3^- =12 \\
& x_1,x_2,x_3^+ ,x_3^- ,s_1,s_2 \geq 0
\end{aligned}
$$

为了让大家更加深入的理解之前引入的x_3^+ ,x_3^-这两个变量，我们去考察一下如下的等价性关系。如图1.1，左边灰色矩形框中的式子是可以推出右侧灰色矩形框的式子的。这在之前我们x_3分别取值为4和-4的实例中就可以看到。那么右边的式子是否可以反推得到左边的式子呢？我们来看图1.2的证明

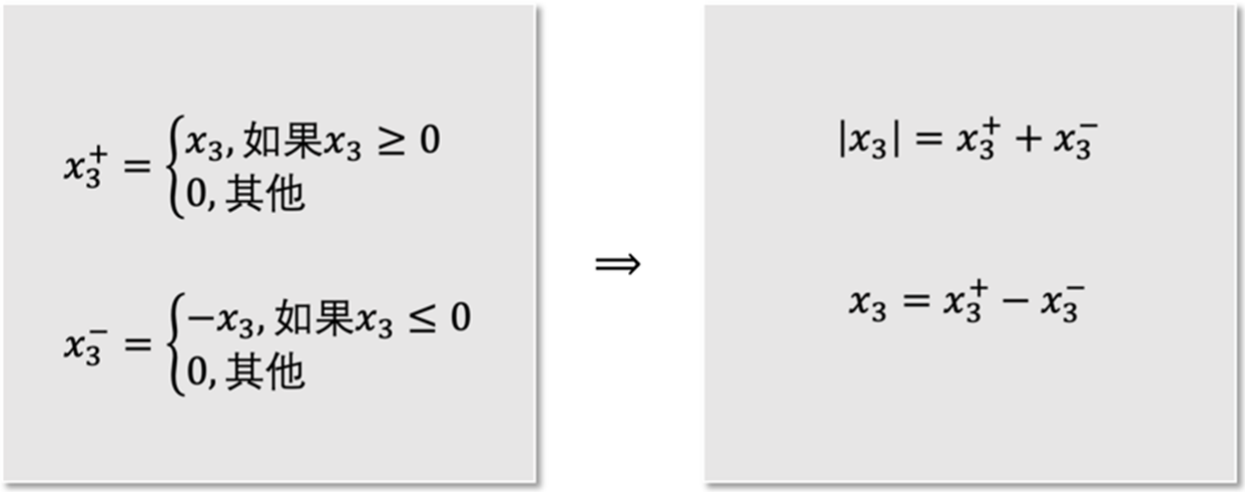


图1.1 “等价性”说明1

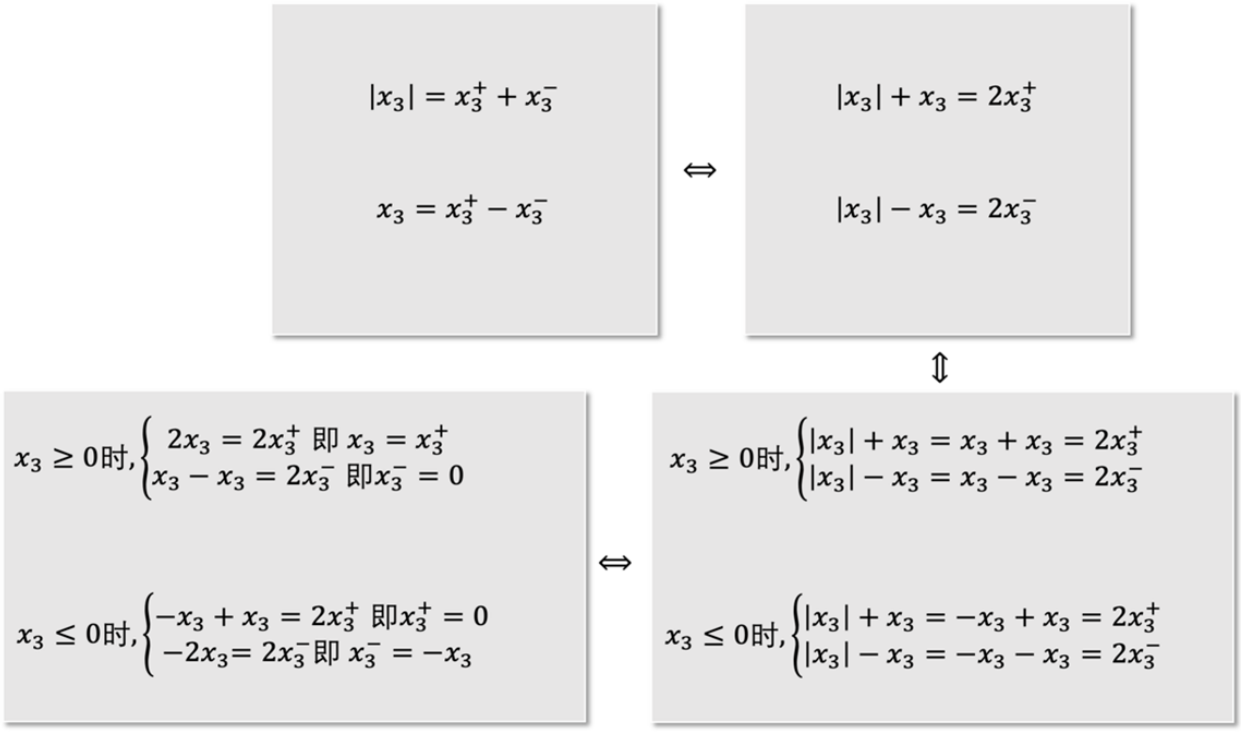


图1.2 “等价性”说明2

那么，之前定义的x_3^+, x_3^-是不是就完美无缺了呢？并不是。

请看下面这个线性规划模型：

$$\begin{aligned} \min \quad & x_3 \\ \text{s.t.} \quad & |x_3| \leq 10 \\ & x_3 \geq -2 \end{aligned}$$

其中并没有非负约束。这个问题的最优解极其明显，就是$x_3 = -2$。

我们将$x_3 = x_3^+ - x_3^-, |x_3| = x_3^+ + x_3^-$代入到上面的例子中，就有：

$$\begin{aligned} \min \quad & x_3^+ - x_3^- \\ \text{s.t.} \quad & x_3^+ - x_3^- \leq 10 \\ & x_3^+ - x_3^- \geq -2 \\ & x_3^+, x_3^- \geq 0 \end{aligned}$$

这个问题读者现在可能不会求解，但是我们可以告诉读者。这个问题的最优解是$x_3^+ = 4$，$x_3^- = 6$，$x_3^+ - x_3^- = 4 - 6 = -2$。与原问题确实等价。但是，细心的读者会发现，此时计算$x_3^+ x_3^- = 4 + 6 = 10 \neq |-2| = 2$。原因是，由$x_3^+, x_3^-$的定义可以知道，$x_3^+$,$x_3^-$不能同时取正数，至少有一个为0 (当$x_3 = 0$时，$x_3^+$与$x_3^-$同时取值为零)。所以说，$x_3^+, x_3^-$的取值分别为0和2才是一个完全符合定义的最优解。

那么，为什么会出现这样的情况呢？实际是有一个隐含的二次约束(quadratic constraint)被忽略掉了，即$x_3^+ \times x_3^- = 0$，如果有了它，那么得到的x_3^+, x_3^-就会满足“至少有一个为0”的条件。读者会问，这有什么用吗，我求出来$x_3^+ = 4$，$x_3^- = 6$，还是能得到最优解是$x_3 = -2$。事实上，大多数情况下，确实没必要苛求“至少有一个为0”的条件。所以通常转化中，我们也不要求非要引入这样的二次约束。

1.4线性规划案例

1.4.1 资源分配生产问题

某军工厂生产甲、乙、丙三种产品，生产三种产品需要A、B两种资源，其单位需求量及利润由下表1给出，问每天生产甲、乙、丙三种产品各多少，可使总利润最大？

表1.1 资源分配生成问题信息

甲		乙	丙	资源的最大量
2	A	3	1	100kg
3	B	3	2	120kg
40元	利润	45元	24元	

这个问题的解决需要建立线性规划模型，用三个变量x_1, x_2, x_3代表每天生产甲、乙、丙产品的数量（假设决策变量）。题干提到要使“总利润最大”，所以建立目标函数：

$$z = 40x_1 + 45x_2 + 24x_3$$

目标函数的含义就是总利润的大小，那么函数值自然越大越好，但是需要考虑题干中提到的各种限制，也就是约束条件。因此，可以写出如下的约束条件：

$$\begin{cases} 2x_1 + 3x_2 + x_3 \leq 100 & \text{使用A资源不能超过给定总量} \\ 3x_1 + 3x_2 + 2x_3 \leq 120 & \text{使用B资源不能超过给定总量} \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 & \text{甲、乙、丙三种产品数量是非负数} \end{cases}$$

经过三个步骤：假设决策变量、建立目标函数、寻找约束条件，我们就得到了如下的线性规划模型：

$$
\begin{aligned}
\max \quad & z = 40x_1 + 45x_2 + 24x_3 \\
\text{s.t.} \quad & 2x_1 + 3x_2 + x_3 \leq 100 \\
& 3x_1 + 3x_2 + 2x_3 \leq 120 \\
& x_1 \geq 0, x_2 \geq 0, x_3 \geq 0
\end{aligned}
$$

只要解出满足这个模型的约束同时使目标函数最大的x_1, x_2, x_3即为最终答案。

1.4.2 营养问题

一个健身人士需要购买A、B两种食品，已知食品含有营养成份1、2、3的数量及每日人体对该三种成份的必需量如表2所示，请问他应当每天分别购买多少A、B食品，使得在满足要求的情况下总费用最少？

表1.2 营养信息

	A	B	每日该成份最低摄入量
成份1	10	4	20mg
成份2	5	5	20mg
成份3	2	6	12mg
食品价格	0.6元	1元	

本问题仿照例一，应当先假设决策变量，设x_1为每天购买食品A的量，设x_2为每天购买食品B的量。 下一步为建立目标函数，这个题目要求我们尽可能让健身人士少花钱，也就是总花费最小，所以有:$\min z = 0.6x_1 + x_2$ 。 最后一步就是考虑约束条件，本例要求每天摄入三种营养成分的数量应至少达到表1.2中要求的最低摄入量，所以有：

$$
\begin{cases}
10x_1 + 4x_2 \geq 20 & \text{成份1需满足最低摄入量要求} \\
5x_1 + 5x_2 \geq 20 & \text{成份2需满足最低摄入量要求} \\
2x_1 + 6x_2 \geq 12 & \text{成份3需满足最低摄入量要求} \\
x_1 \geq 0, x_2 \geq 0 & \text{A,B两种食品数量是非负数}
\end{cases}
$$

所以得到如下的线性规划模型：

$$
\begin{aligned}
\min \quad & z = 0.6x_1 + x_2 \\
\text{s.t.} \quad & 10x_1 + 4x_2 \geq 20 \\
& 5x_1 + 5x_2 \geq 20 \\
& 2x_1 + 6x_2 \geq 12 \\
& x_1 \geq 0, x_2 \geq 0
\end{aligned}
$$

同理，只要解出满足这个模型的约束同时使目标函数最小即总花费最小的x_1, x_2即为最终答案。

1.4.3卷钢问题

卷钢问题又称为下料问题，这个问题严格来讲不能算是“线性规划”的内容，而是“整数规划”的内容，所谓整数规划，读者可以理解为是在线性规划的基础上，给决策变量加上整数限制，即决策变量必须是整数，这比线性规划的求解难度要大得多。但是这个问题在建模的角度上又是一个非常典型的问题，所以特此介绍。

本例采用卷钢问题中最简单的一种，也就是一维卷钢问题。 有一个卷钢加工工厂，现在有100个卷钢，每个卷钢(roll)的宽度都是1000。 现在有5个客户，每个客户都需要一定数量的零件(item)，这些零件都可以通过卷钢切割加以生产。 比如说1号客户需要5个宽度为100的零件，那么工厂可以按照图1.3的方式进行生产。 2号客户需要4个宽度为150的零件；3号客户需要14个宽度为375的零件；4号客户需要7个宽度为800的零件；5号客户需要3个宽度为50的零件。

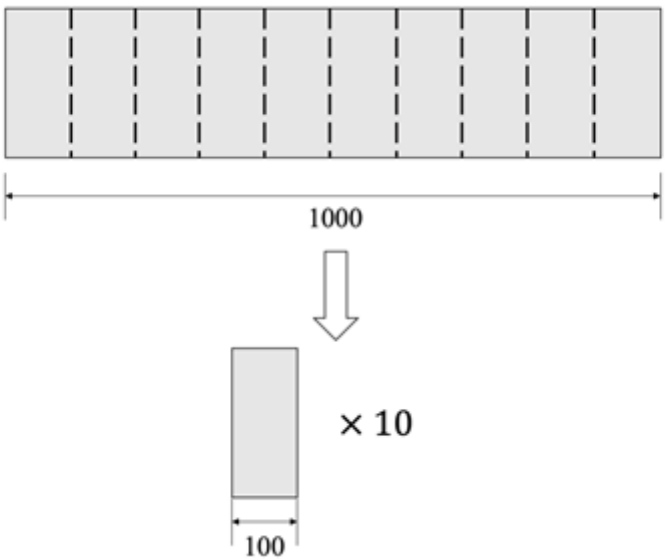


图1.3 一种切割方式

现在的问题是，如何安排使用卷钢数量最少的切割计划，使得所有客户的需求能够满足，同时还要考虑到卷钢的宽度是有限制的。

下面我们建立问题的模型，第一步为假设决策变量。 考虑到工厂一共有1000个卷钢，面对5个客户的需求，不一定所有的卷钢都要得到使用，所以我们引入一种取值仅为0或1的决策变量 y_k，其中$k=1,\cdots,1000$表示第k个卷钢。 如果y_k 取1，表示 k号卷钢被用来切割零件；如果 取0，表示 号卷钢没有被用来切割零件。 题干要求我们安排使用卷钢数量最少的切割计划，因此我们的目标函数其实就是$\sum_{k=1}^{1000} y_k$。除了y_k，我们还要引入一个决策变量，它也是取值为非负整数，但是不局限于0和1，而是所有非负整数，它就是x_i^k, 其中$i=1,\cdots,5$表示第i位客户，或者说第i种零件。 这个决策变量的含义是在k号卷钢上切下第i种零件的个数。 由于目标函数已经建立完成，所以建模只剩下考虑约束条件了。 第一类约束条件是所有客户的需求要得到满足，这里有5个客户，所以有5条约束，我们用符号n_i表示客户i需要的零件个数，那么我们就有$sum_{k=1}^{1000} x_i^k \geq n_i, i=1,\cdots,5$。第二类约束条件是在每个卷钢上的切割方案需要考虑到卷钢总宽度的限制，但是不一定1000个卷钢都要拿来投入生产用以满足这5位客户的要求，也就是说有一些卷钢不会进行切割，那么如何表达这一事实呢，我们其实可以引入刚才使用的决策变量 y_k，那么约束就为:
$\sum_{i=1}^{5} w_i x_i^k \leq W y_k$，这一类约束共有1000个，其中$w_i$为零件$i$的宽度。 综上，卷钢问题的模型如下所示：

$$\min \quad \sum_{k=1}^{1000} y_k$$

$$\text{s.t.} \quad \sum_{k=1}^{1000} x_i^k \geq n_i, \quad i=1,\cdots,5$$

$$\sum_{i=1}^{5} w_i x_i^k \leq W y_k, \quad k=1,\cdots,1000$$

$$x_i^k \in \mathbb{Z}^+ \cup \{0\}, y_k \in \{0,1\}, \quad i=1,\cdots,5, k=1,\cdots,1000$$

该问题实际上并不是线性规划，因为决策变量加上了必须为整数这一约束，这其实是整数规划问题，这个问题的模型最早是前苏联的学者Kantorovich建立的。 这个模型虽然非常清晰地描述了一维卷钢问题，但是直接求解这个模型效率非常低，后来Gomory针对此问题建立了其他版本的模型，并应用著名的列生成算法(column generation)进行高效求解，这些内容就不在这里展开描述了，读者感兴趣的话可以阅读本书第5章列列生成算法部分。

本章参考文献

[1] 马国瑜. 线性规划的发展历史[J]. 北京化工大学学报(自然科学版), 1985(4):27-33.

[2] SC Fang. Linear Optimization and Extensions – Theory and Algorithms[M]. Prentice-Hall, Inc,1993, 1-9.

第2章 线性规划的几何原理

作者：孙睿晗，荷兰阿姆斯特丹大学 人工智能硕士研究生在读

研究方向：强化学习，元学习，贝叶斯优化

本章节，我们将从几何的角度认识线性规划问题，主要介绍凸集，极点，极方向，空间多面体，超平面等概念。

2.1 凸集

2.1.1 凸集定义

如果一个集合C是凸集,那么我们可以说，对于集合内任意两点x、y，$z = \theta x +（1-\theta)y$也在$C$里面。注意到，当$\theta$等于0的时候，$z = y$, 当$\theta$等于1的时候, $z = x$, 也就是说从几何角度来理解z是一条连接x和y的线段。从这种角度来理解凸集，我们可以想象对于凸集内任意两点的连线仍在此凸集内。如下两幅图可以很直观的说明这个概念。

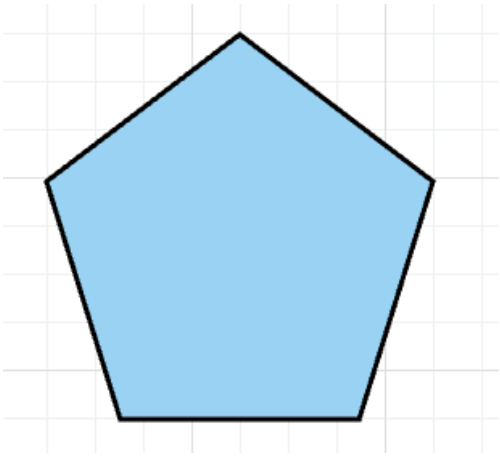


图2.1 正五边形

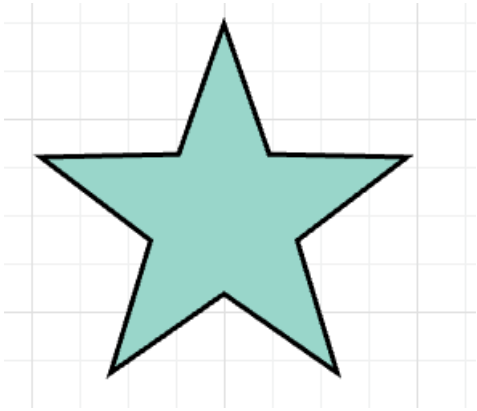


图2.2 五角星

对于图2.2五角星来说，存在两点，它们之间的连线并不在五角星内，所以五角星不是凸集。而图2.1正五边形则很明显是一个凸集。

2.1.2 Lemma引理

已知S_1, S_2是两个凸集，那么：

$S_1 \cap S_2$是凸集

$S_1 \oplus S_2 = \{x_1 + x_2 : x_1 \in S_1, x_2 \in S_2\}$是凸集

$S_1 \ominus S_2 = \{x_1 - x_2 : x_1 \in S_1, x_2 \in S_2\}$是凸集

2.1.3 凸壳（Convex Hull）

为了定义凸壳，首先我们需要知道什么是凸组合（convex combination），类似于凸集的定义，凸组合定义如下：

$$x = \sum_k \lambda_k x_k$$

$$\sum_k \lambda_k = 1$$

$$\lambda_k \geq 0, \forall k$$

也就是说只要变量前的系数相加为1且都为非负，我们就有了一个凸组合。现在，我们可以定义凸壳了。

凸壳定义：对于一个集合$S \in R^n$, S的凸壳$conv(S)$是S内所有元素的凸组合的集合。即如果$x \in conv(S)$,那么x可以表示为S内k个元素的凸组合：

$$x = \sum_k \lambda_k x_k$$
$$\sum_k \lambda_k = 1$$
$$\lambda_k \geq 0, \forall k$$
$$x_k \in S, \forall k$$

2.2 空间多面体

2.2.1 空间多面体(Polyhedron)

因为空间内任何线性不等式$a_j^T x \leq b_j$可以将空间分为两个部分，所以我们统称线性不等式为**halfspaces**。同样对于任意线性等式$a_q^T x = b_q$可以在空间内定义一个“平面”，这在三维空间很好想象，因为它就是一个平面的解析式，但在高维空间我们叫“平面”为“超平面”。则空间多面体就是有限个**halfspaces**和超平面的相交部分，数学语言来表示为：

$$P = \{\boldsymbol{x} | \boldsymbol{a_j^T x} \leq b_j, j = 1, \ldots, p; \boldsymbol{a_q^T x} = b_q, q = 1, \ldots, m\}$$

我们还可以把P写成矩阵-向量的形式：

$$P = \{\boldsymbol{x} | A\boldsymbol{x} \leq \boldsymbol{b}, C\boldsymbol{x} = \boldsymbol{d}\}$$

这里的x, b, d均为向量，A和C是两个矩阵。我们可以证明任何空间多面体都是凸集，只需要在P中任取两点x和y，然后证明x, y的凸组合（convex combination）$z \in P$即可，在此不赘述。

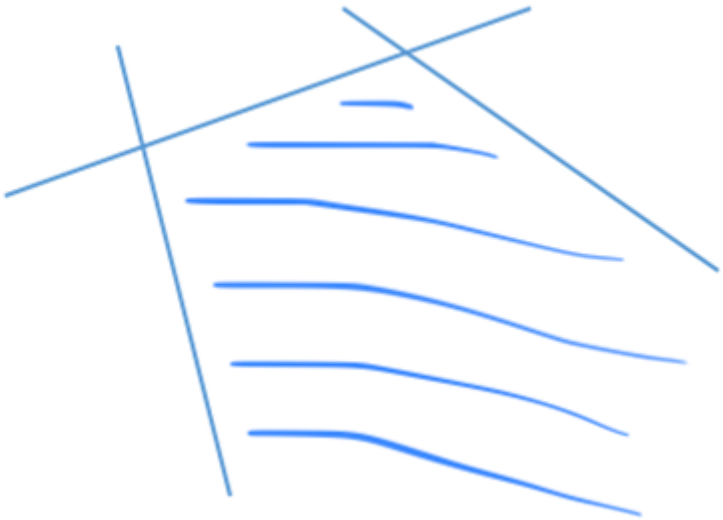


图2.3 二维空间的空间多面体（阴影部分）

2.2.2 多面体和锥(Polytopes and Cones)

多面体就是一个有界的空间多面体，也就是说多面体是空间多面体的一个特殊情况，空间多面体会出现有界限、无限延伸的情况，但多面体不会（见图2.4）。

锥，顾名思义，几何上理解就是一个没有棱的圆形金字塔（见图2.5），接下来我们谈谈锥的数学定义。

定义：如果$x \in C$，对于任何 $\lambda \geq 0$，$\lambda x \in C$。

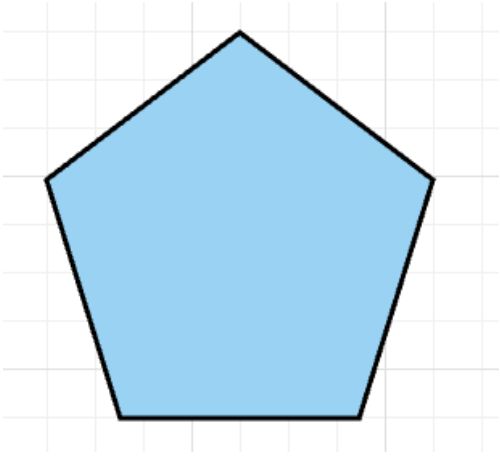


图2.4 多面体为有界空间多面体

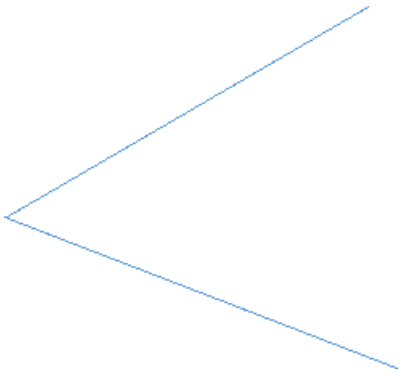


图2.5 两条从原点引出的射线组成一个锥

类似于凸集有凸组合(convex combination)，锥也有锥组合(conic combination)。k个n维向量x_k的锥组合可以写为：

$$Z = \lambda_k x_k, \lambda_k \geq 0, \forall k$$

注意在这里我们并不要求λ为0到1之间的数，因为我们需要保留“锥”的特征。所有锥内部元素的锥组合所构成的集合我们称之为凸锥(convex cone)，从名字我们不难猜测，凸锥的定义就是它即是一个锥，又是一个凸集。如果x在凸集C中，那么x所有的锥组合仍属于C，反之，如果C是凸锥，那么C 包含其所有元素的锥组合。这里我们需要将凸锥的概念与凸壳**(convex hull)**区分开来，凸壳是凸锥的一个子集。凸锥包含锥所有元素的锥组合，而凸壳则刚刚好等于锥所有元素的锥组合。

2.3 极点与极方向

2.3.1 极点与极方向

在这一部分我们来谈一谈极点的概念，首先直观上理解极点就是空间多面体的“顶点”，其数学定义有一些抽象，但与我们的直观理解并不冲突。

定义：对于任意凸集S，S内一向量$\boldsymbol{x}$如果是S的极点，那么不存在不同于$\boldsymbol{x}$的两个向量 $\boldsymbol{y}, \boldsymbol{z}$，使得$\boldsymbol{x}=\lambda \boldsymbol{y}+(1-\lambda) \boldsymbol{z}, 0 \leq \lambda \leq 1$。

有了极点的定义后，我们来定义凸集的极方向。首先我们考虑什么是凸集的“方向”，通俗一点讲就是给定一个S内的起始点，然后沿着这个方向无论走多远我们依旧会落在原来的凸集S内。而极方向的定义我们可以和极点联系起来，简单来说就是极方向不能是凸集内另外两个不同方向的锥组合。这里的锥组合就是线性组合，只不过系数均为正，而$\boldsymbol{a}, \boldsymbol{b}$不同方向意味着$\boldsymbol{a} \neq k \boldsymbol{b}$。如果把在凸集中极方向的定义映射回几何学中，就是空间多面体的边，但注意应是无限延长的边，对于那些有限长度的边，并不符合极方向的定义。

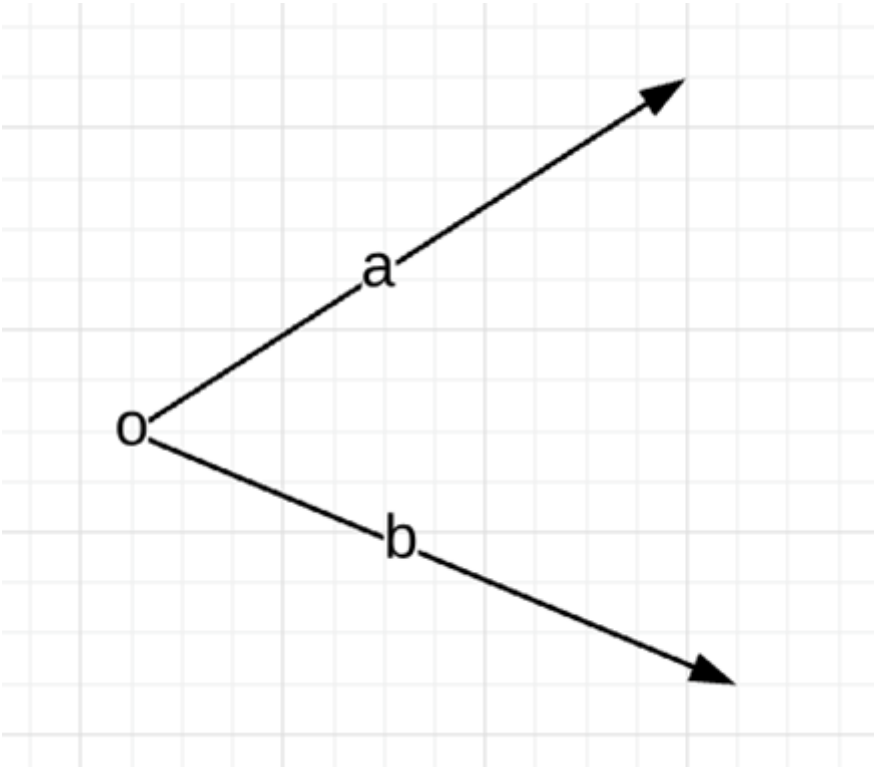


图2.6 o为极点，a,b为锥的两个极方向

2.3.2 极点的特性

对于一个空间多面体P, 已知$x^* \in P$, 那么以下三个条件为等价条件：

x^*是 P 的一个顶点(vertex)。

x^*是 P 的一个极点(extreme point)。

x^*是P的一个基本可行解(basic feasible solution)。

证明这个定理的思路是先证明$1 \Rightarrow 2$,然后证 $2 \Rightarrow 3$,最后证 $3 \Rightarrow 1$，通过这么一个循环，我们可以得到1，2，3为等价条件。我将给出$1 \Rightarrow 2$的简要证明思路，有兴趣的读者可以查询Bertsimas. et al[1]。首先让我们定义什么是空间多面体的顶点，如果x是P 的顶点，那么对于P内任意不同于x的点y，一定存在c使得$cx < cy$。现在让我们假设$x^* \in P$是P的顶点，那么一定存在$y \neq x, z \neq x \quad s.t. cx < cy, cx < cz$。整理一下可得$cx < c(y + z)$。又可以知道对于$0 \leq \lambda \leq 1$，$\lambda cx < \lambda cy, (1 - \lambda)cx < (1 - \lambda)cz \Rightarrow cx < c(\lambda y + (1 - \lambda)z)$。即$x \neq \lambda y + (1 - \lambda)z$, 也就是说$x$是$P$的极点，证毕。

这个定理对我们的意义在于求解线性规划问题可以直接找空间多面体的顶点，因此省去了很多不必要的麻烦。其实后期可以证明线性问题的至少一个最优解一定会出现在顶点上，这也是单纯形法(simplex method)背后的主要思想。

2.4 超平面

超平面定理（**Hyperplane Theorem**）

Hyperplane Theorem是凸集中最基本的定理之一，其主要内容如下：

定理 **1**：$C, D \in R^n$ 为两个不相交的凸集,那么存在$a \in R^n, a \neq 0, b \in R$可以使得$\forall x \in C, a^T x \leq b$以及$\forall x \in D, a^T x \geq b$。

定理 **2**：$C, D \in R^n$ 为两个不相交的闭凸集,那么存在$a \in R^n, a \neq 0, b \in R$可以使得$\forall x \in C, a^T x < b$以及$\forall x \in D, a^T x > b$。

定理2是定理1的一个特殊情况。定理2中这条可以将C, D分开的直线可以简单定义为C, D的垂直平分线。

本章参考文献

[1] Dimitris Bertsimas and John Tsitsiklis. Introduction to Linear Optimization.Athena Scientific, 1st edition, 1997.

[2] Mokhtar S. Bazaraa. Nonlinear Programming: Theory and Algorithms. Wiley Publishing, 3rd edition, 2013.

[3] Stephen Boyd and Lieven Vandenberghe. Convex Optimization. Cambridge University Press, New York, NY, USA, 2004.

第3章 单纯形法

作者：臧永森，清华大学 工业工程系 在读博士，戚铭尧老师团队

研究方向：运筹优化算法设计与应用、数据统计分析、大数据技术与应用、鲁棒优化

通过前面两章的介绍，相信大家已经对线性规划有了比较深入的理解，本章将介绍一种求解线性规划比较经典的算法，即单纯形法。单纯形法由美国数学家George Bernard Dantzig在1947年担任美国空军司令部数学顾问时提出，旨在解决空军军事规划问题，之后成为解决线性规划问题比较通用的有效算法。Dantzig在1953年又提出了改进单纯形法，目的是降低迭代过程中的累积误差；美国数学家Carlton Edward Lemke于1954年提出对偶单纯形法，来解决原问题难以求解，但是对偶问题方便求解的问题。这些变形单纯形法都基于基础单纯形法，本章主要介绍基础的单纯形法。

3.1 可行域与最优解

谈线性规划问题目的就是要进行求解获取最优解，要理解最优解就不得不提可行域，可行域和最优解是线性规划问题中最为重要的两个概念。

线性规划问题的可行域指的是满足线性规划问题所有约束条件（包含符号约束等各种约束）的所有点的集合，每个点叫做可行解。最优解是指可行域中使得目标函数取得最小值（对于最小化问题）或最大值（对于最大化问题）的某一个或某些点，相应的函数值就是最优值。

用一个比较生动的例子来解释上述两个概念：假设某个周末小明来到采摘园享受轻松时光，采摘园有三块相邻的采摘区，分别属于三家农户：老张、老李、老王，老张种了草莓，老李种植的西瓜，老王则种植了葡萄，虽然三块采摘区相邻，但三家农户分别做自己的生意互不相往来。小明决定到老李家采摘西瓜，老李做生意的规定是：每位顾客只能进入园区一次，只能吃或者带走一个西瓜，进入园区的门票是20元人民币。

当小明进入老李的西瓜园区后，就可以来分析我们的可行域和最优解了。虽然老李家的西瓜园区与隔壁的草莓园区、葡萄园区相邻，但是小明是不可以进入草莓园区和葡萄园区进行采摘的，因为他买的是老李家的门票，而草莓园区和葡萄园区不属于老李，如果进入草莓园区或者葡萄园区，老张或者老王就要找小明或者老李讨说法了。也就是说，小明只能在西瓜园区进行采摘，那么西瓜园区就是小明的采摘范围，这个采摘范围就可以看成是小明活动的可行域。

那最优解对于小明来说是啥呢？小明花了20元钱进入采摘园，老李规定他只能享用一个西瓜，那么正常情况下小明会选择一个最大又最熟的西瓜来吃或者带走，这样才会使得进入园区的利益最大化，也就是我们俗话说的“对得起那20块钱”。而西瓜园区内成百上千个西瓜都是小明可以选择的，这些可以选择的西瓜可以看成是小明的可行解，而最大的最熟的那个就是最优解。当然，如何从所有西瓜中找到最大最熟的那一个，就是如何求解线性规划问题的研究范围了。

把上述场景简化并抽象到坐标系中，就是一个简单的线性规划问题模型，令x、y分别代表采摘园区的长度和宽度，水果在园区的位置对应相应的坐标点，假设西瓜大小与成熟度和坐标点存在某种线性关系，那么模型可以粗略表示如下图3.1所示：

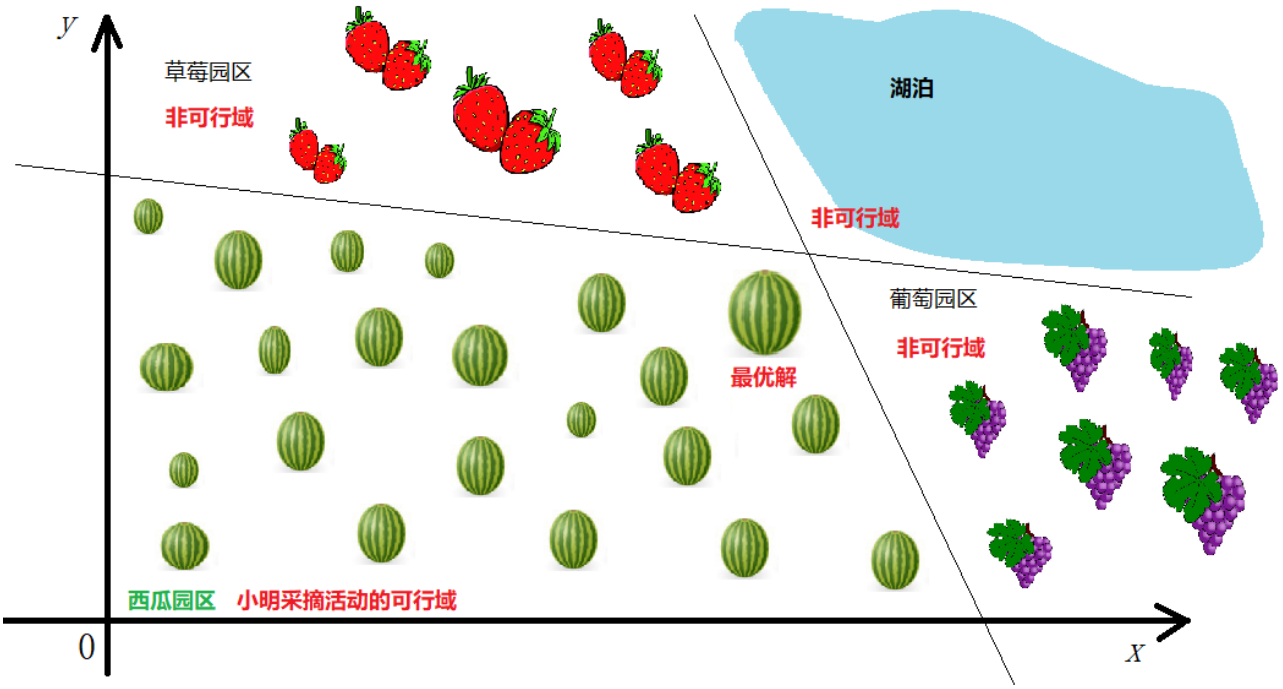


图3.1 可行域与最优解通俗解释图示

把上述例子继续抽象简化，将其设计为精确的线性规划问题，可以抽象为如下模型：

$$\max \quad z = x + y$$

$$\text{s.t.} \quad x + 2y \leq 24$$

$$2x + y \leq 30$$

$$x \geq 0, y \geq 0$$

该模型的图示，如图**3.2**所示，其中蓝色填充区域为该模型的可行域，红色直线表示模型的目标函数，绿色剪头表示红线向上移动能够使得目标函数值增大，移动的过程中不能超出可行域（蓝色区域），因此能够使目标函数取得最大值的点，就是黑色标注点（12,6）这一点，该点就是该模型的最优解，相应的最优值为$z = 12 + 6 = 18$.

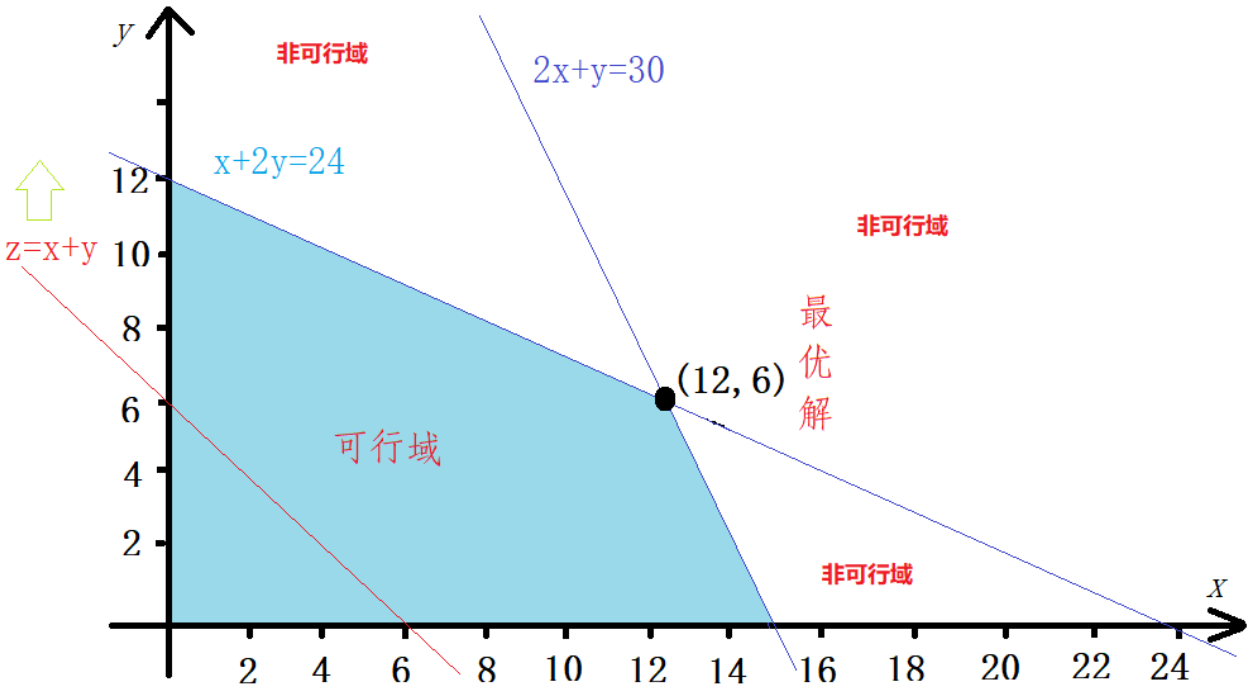


图3.2 模型可行域与最优解精确解释图示

3.2 基变量与基可行解

除了上述两个概念，在线性规划问题的单纯形法求解过程中，还绕不开基变量和基可行解两个重要概念，下面通过线性方程组的知识引入这两个概念。

考虑n个变量、m个线性方程所构建的方程组$A\mathbf{x} = \mathbf{b}$，假设$n \geq m$且A有$m \times m$的满秩子矩阵（该子矩阵称为基），令$n - m$个变量等于0，然后根据克莱姆法则可知能够求出满足$A\mathbf{x} = \mathbf{b}$的其余m个变量的唯一值。此时这$n - m$个变量就是非基变量，其余m个变量就是基变量。而由上述基变量和非基变量组成的变量叫做基解。

考虑由$A\mathbf{x} = \mathbf{b}$组成的线性规划模型：

$$\min \quad z = \sum_{i=1}^{n} c_i x_i \tag{3.1}$$

$$\text{s.t.} \quad A\mathbf{x} = \mathbf{b}(A是m \times n矩阵) \tag{3.2}$$

$$x_i \geq 0(i = 1, 2, .., n) \tag{3.3}$$

如果上述讨论的线性方程组$A\mathbf{x}=\mathbf{b}$的基解满足线性规划模型中的符号约束(3.3)，那么它就是基可行解。

在上述分析中，我们是任意令$n-m$个变量为0，这也就意味着，$n-m$个变量的选择是具有不确定性的，基于此求得的基变量就不同，也就是说在解的集合中会出现不同的非基变量与基变量组合而成的基解。

下面我们通过一个简单的例子理解上述概念，考虑下述线性规划问题：

$$\begin{aligned}
\max \quad & z=2x_1+3x_2+x_3 \\
\text{s.t.} \quad & x_1+x_3=5 \\
& x_1+2x_2+x_4=10 \\
& x_2+x_5=4 \\
& x_1,x_2,x_3,x_4,x_5\geq 0
\end{aligned}$$

该线性规划问题涉及的线性方程组$A\mathbf{x}=\mathbf{b}$中的A为3×5的矩阵

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

容易看出矩阵

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

是线性规划问题的一个满秩矩阵，因此该矩阵的秩为3，令5-3=2个变量等于0，任意取$x_1=0,x_4=0$（这里的任意是在保证满足方程组本身方程等式的条件下取值，比如我们不能取$x_1=0,x_3=0$，因为这样就违背了方程$x_1+x_3=5$），此时求解$A\mathbf{x}=\mathbf{b}$可以求得$x_2=5,x_3=5,x_5=-1$，对应的列向量组成的矩阵

$$\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

是线性规划问题的一个基，其中的三个列向量均称为基向量，与其对应的x_2, x_3, x_5就是基变量，而x_1, x_4就是非基变量。它们组成的变量
$X = (x_1 = 0, x_2 = 5, x_3 = 5, x_4 = 0, x_5 = -1)$叫做基解，此时$x_5 = -1$不满足线性规划的约束$x_5 \geq 0$，因此该基解不是基可行解。

为了帮助读者更好地理解基可行解的概念，我们再考虑任意令$x_4 = 0, x_5 = 0$，此时求解可$A\mathbf{x} = \mathbf{b}$以求得$x_1 = 2.x_2 = 4, x_3 = 3$，那么x_1, x_2, x_3就组成了基变量，而x_4, x_5就是非基变量，它们的组合$X = (x_1 = 2, x_2 = 4, x_3 = 3, x_4 = 0, x_5 = 0)$就是基解，同时基解满足$x_1, x_2, x_3, x_4, x_5 \geq 0$，因此该基解也就是基可行解。其实还可以验证，没有其它基可行解对应的目标函数值大于该基可行解对应的函数值，即该基可行解为该线性规划的最优解，将其带入线性规划的目标函数可得最优值为$z = 19$。

为了更加清晰地表示基解和基可行解以及可行解之间的关系，将例子中的全部基解以及部分可行解列于表3.1：

表3.1 例题中基解、基可行解以及可行解之间的关系

序号	**X1**	**X2**	**X3**	**X4**	**X5**	目标函数值**Z**	是否可行解	是否基解	是否基可行解
1	0	0	5	10	4	5	是	是	是
2	0	4	5	2	0	17	是	是	是
3	5	0	0	5	4	10	是	是	是
4	5	2.5	0	0	1.5	17.5	是	是	是
5	2	4	3	0	0	19*	是	是	是
6	0	5	5	0	-1	20	否	是	否
7	10	0	-5	0	4	15	否	是	否
8	5	4	0	-3	0	22	否	是	否
9	2	2	3	4	2	13	是	否	否
10	1	1	4	7	3	11	是	否	否
11	3	3	2	1	1	17	是	否	否

从上表可以很清晰地看出，可行解可能是基解，也可能是非基解，可行解中的基解便是基可行解；基解中包含可行解和非可行解，基解中的可行解就是基可行解。因此，基可行解必须同时是可行解和基解。

3.3 单纯形法的基本思想

前两节我们为引入单纯形法介绍了可行域、最优解、可行解、基解、基可行解等基础概念，也阐述了它们之间的相互关系。在明确了这些基本概念之后，这一节我们来探讨单纯形法的思想逻辑和求解步骤。

3.3.1 基本思想和逻辑

上一节我们已经知道，优化问题的最优解一定是基可行解，那么如何找到最优的基可行解就是最优化问题的求解思路。因此，单纯形法的求解过程，就是不断地寻求变量出入基的循环迭代过程，每次迭代都达到降低目标函数值（或增大目标函数值）的目的，最终得到最优解。那么在迭代过程中，如何使解在改善过程中向着最优解的方向尽快地收敛呢？我们下面用比较直观的方式来解析这个过程。

本文采用的思路参考Dimitris Bertsimas和 John N. Tsitsiklis在 Introduction to Linear Optimization一书中提出的方法[2]，考虑如下标准线性规划问题：

$$\min \quad z = \mathbf{c}^T\mathbf{x} \tag{3.4}$$

$$\text{s.t.} \quad A\mathbf{x} = \mathbf{b}(A是m \times n矩阵) \tag{3.5}$$

$$\mathbf{e}^T\mathbf{x} = 1(\mathbf{e}是n维列向量) \tag{3.6}$$

$$x_i \geq 0(i = 1, 2, .., n) \tag{3.7}$$

我们将矩阵A拆分为n个列元素：$A_1, A_2, \ldots, A_n$，那么我们可以将问题看成是满足非负约束(3.7)、凸约束(3.6)以及约束(3.8)的最小化问题。

$$\begin{bmatrix} A_1 \\ c_1 \end{bmatrix} x_1 + \begin{bmatrix} A_2 \\ c_2 \end{bmatrix} x_2 + .. + \begin{bmatrix} A_n \\ c_n \end{bmatrix} x_n = \begin{bmatrix} b \\ z \end{bmatrix} \tag{3.8}$$

结合式(3.6)和(3.8)我们可以看出，原优化问题转化为求解能够构造出(b, z)的使得z值最小的关于(A_i, c_i)的凸组合。为了更好地理解它们之间的几何关系，我们将一个平面视作包含A的一个m维空间，将与c_i相关的成本项看作是一维垂直数轴，这时每一个点(A_i, c_i)都可以唯一在该三维坐标系中表示出来，如图**3.3**所示：

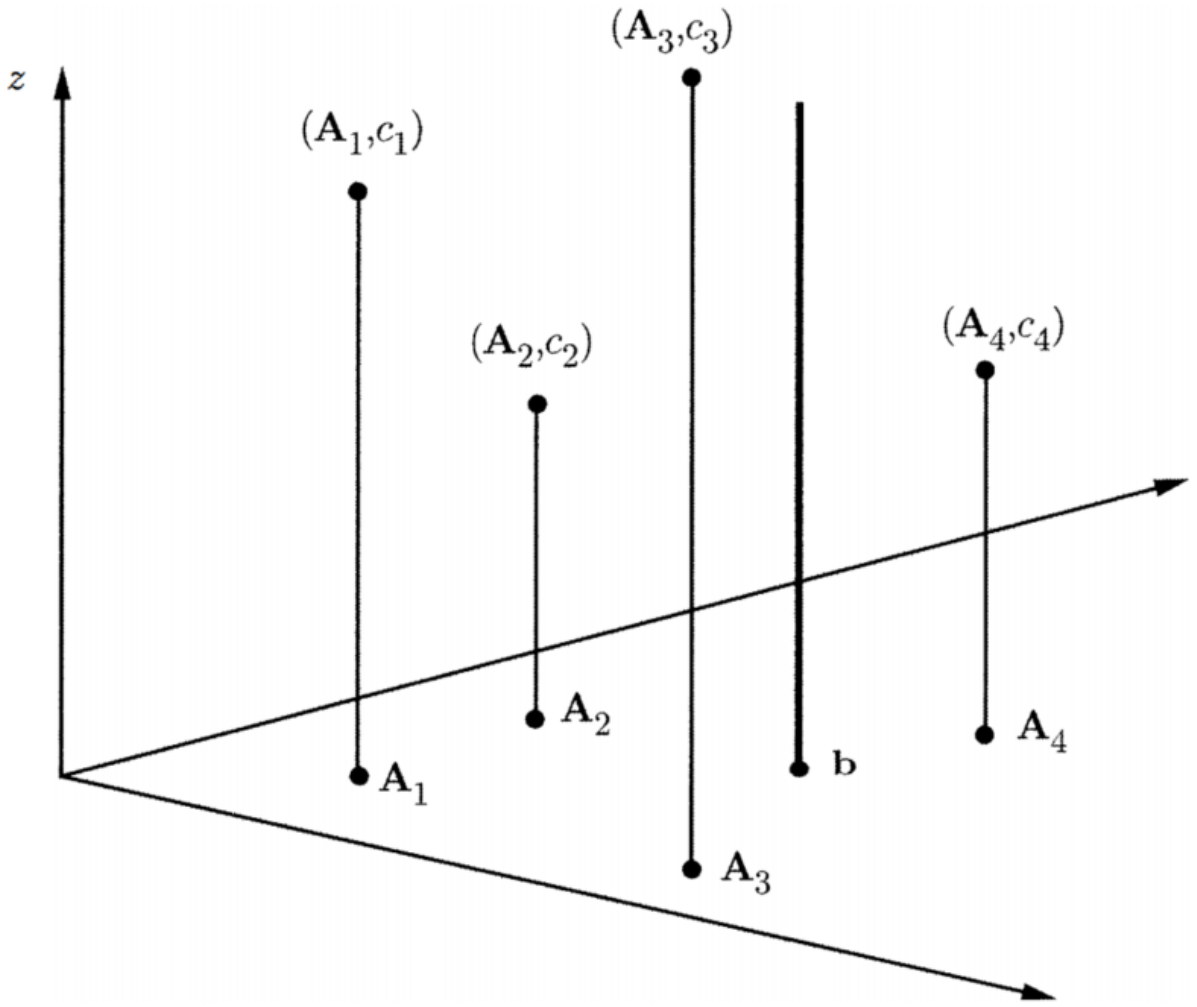


图3.3 线性规划问题3.5—3.8的“列几何”图示

我们将(b,z)同样视为一条垂直线表示在图**3.3**中，这条垂直线叫做需求线，其与平面的交点是$(b,0)$点。需求线与(A_i,c_i)的凸组合在几何上有一定的关系，它们或相交或相离，这取决于我们对(A_i,c_i)凸组合的选取，选取的凸组合不一样，几何关系就不同。很容易能理解，如果需求线和凸组合相交，说明(b,z)可以用相应的凸组合表示出来，也就表明这个凸组合就是原问题的一个可行解；而如果相离，则说明这个凸组合不满足能够表达(b,z)的条件，也就不是原问题的可行解。所有的凸组合构成了一个凸包，如果需求线能够与凸包相交，那么原问题就存在可行解，如果需求线不能与凸包相交，说明原问题无解。进一步将图**3.3**抽象，得到图**3.4**，从图中我们可以看出，点I、H、G就是三个不同的凸组合与需求线的交点，也就是原问题的三个可行解。

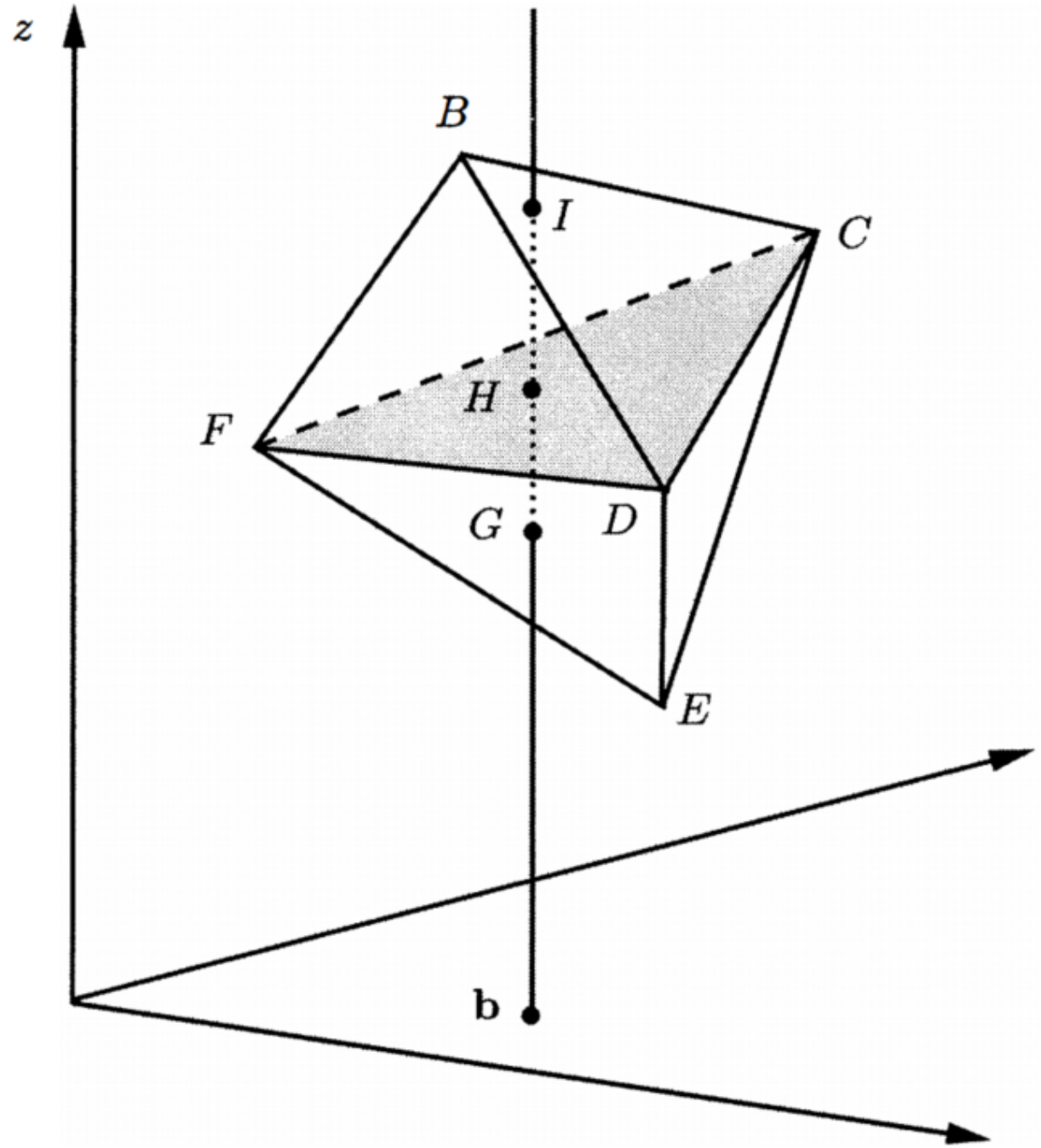


图3.4 可行解的“列几何”图示

经过上面的分析我们得知，要找到最优解，就是找到与需求线相交的使得z值最小的凸组合。那么如何找这样的凸组合呢？首先引入两个定义：

（1） 如果向量$y^1 - y^{k+1}, y^2 - y^{k+1}, \ldots, y^k - y^{k+1}$是线性独立的，那么向量$y^1, y^2, \ldots, y^k, y^{k+1}$被称为$R^n$空间中的仿射独立或者仿射无关。其中$k \leq n$。

（2） 在R^n空间中由$k+1$个仿射无关向量组成的凸包被称为k维单纯形。

对模型(3.4—3.7)来说，总共有$m+1$个等式约束，假定约束系数矩阵是满秩的，那么一个基可行解将对应$m+1$个线性独立的列向量，也就意味着有$m+1$个基点，根据上述定义，由基点之间的差向量线性独立可以得到其仿射独立，由此可以知道它们组成的凸包是m维单纯形。

假设m维单纯形与需求线相交于点(b,z)，由(3.8)知用来表示(b,z)的线性组合的权重向量是x_i，该向量就是一个基可行解，也就对应我们上节所分析的基变量的内容，当然z就是相应的目标函数值。我们用图**3.4**做一个解释，阴影区域的三角形CDF，就是一个二维单纯形，其与需求线的交点H点就是基可行解，点C、D、F是基点。

我们对二维单纯形CDF做一些改变，会发现相应的z值（与需求线的交点）也会变化，比如我们令基点B取代基点F，单纯形变为BCD，这时可行解变为I点，相应的z值较之前有所增长。相类似地，假如用将点E变为基点，单纯形由CDF变为EDF，可行解就出现在G点，此时z值有所减小。从这些变化中我们找出这样一个规律，当且仅当新加入基的点在当前单纯形平面上方（下方）时，所得的交点（即可行解）对应的z值会增大（减小）。

如果我们更加形象地描述这个基点变化的过程，就如同用手抓住单纯形CDF的基点C，保持D点和F点固定不变，用力向上拉（向下拉），将C点拉到B点（E点），也就产生了新的单纯形BCD（EDF）。单纯形法的旋转迭代过程，就是不断找到基点向上拉（向下拉）到新基点形成新单纯形的过程。

3.3.2 单纯形法求解过程

上一小节我们分析了单纯形法的求解原理，总结一下就是先找到一个基可行解，然后从非基解中找一个比较有前途的点入基，替换掉基可行解中有待改善的基点，从而达到改善目标函数的目的，如此重复迭代，直至无法找到可以入基的点。

下面我们用一个例题来演示单纯形法的求解过程。

例题**3.1** 用单纯形法求解如下LP问题：

$$
\begin{aligned}
\max \quad & z = 2x_1 + 3x_2 \\
\text{s.t.} \quad & x_1 + 2x_2 \leq 8 \\
& 4x_1 \leq 16 \\
& 4x_2 \leq 12 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

第一步：将上述LP化解为标准形式，目的是能够在初始单纯形表中很容易地获得初始基可行解。

$$\begin{aligned} \max \quad & z - 2x_1 - 3x_2 = 0 \\ \text{s.t.} \quad & x_1 + 2x_2 + s_1 = 8 \\ & 4x_1 + s_2 = 16 \\ & 4x_2 + s_3 = 12 \\ & x_1, x_2, s_1, s_2, s_3 \geq 0 \end{aligned}$$

第二步，将标准LP列入初始单纯形表，如图3.5：

	z	x_1	x_2	s_1	s_2	s_3	rhs	Basic Varible	Ratio
0行	1	-2	-3	0	0	0	0	z=0	
1行	0	1	2	1	0	0	8	s_1=8	4
2行	0	4	0	0	1	0	16	s_2=16	∞
3行	0	0	4	0	0	1	12	s_3=12	3

图3.5 初始单纯形表

上述单纯形表中可以看出初始基变量是(s_1, s_2, s_3)，从表中找一个能够入基的变量，要求该变量入基后能够使得目标函数值增大量最大。决策变量在第0行的系数看成是这个变量的缩减成本，就是当这个变量增加1时，目标函数z的值将减少的量。比如x_1的系数是-2，就说明当x_1每增加1，z值将减少-2，也就是增加2。因此如果我们要选择能够使目标函数增加量最大的量入基，应该选择第0行中系数最小的负值（读者可以考虑下为什么必须是负值）。因此这里选择x_2入基。

那如何选择出基变量呢？这里我们采用比值法，用右端项的值除以出基变量对应的列系数（红色线框标注），从中选择最小的比值对应的基变量出基。如果不选择最小比值对应的基变量出基，将会导致后面的迭代过程出现负的右端项，相应行的基变量将为负值，这与LP标准型的变量非负约束相违背，因此这种操作是不被允许的。所以，图3.5中的比值优胜者是3，因此s_3出基（蓝色线框标注）。

第三步：通常我们会在x_2所在列与s_3所在行交汇点圈一个圈，也就是元素4。这表示这一点是我们的转轴点，将此元素通过乘除运算变为1（所在行同时作相同变化），同一列其它元素全部通过行加减运算换算为0，得到下一个单纯形表，如图3.6所示：

	z	x_1	x_2	s_1	s_2	s_3	rhs	Basic Varible	Ratio
0行	1	-2	-3	0	0	0	0	z=0	
1行	0	1	2	1	0	0	8	s_1=8	4
2行	0	4	0	0	1	0	16	s_2=16	∞
3行	0	0	4	0	0	1	12	s_3=12	3
0行	1	-2	0	0	0	0.75	9	z=9	
1行	0	1	0	1	0	-0.5	2	s_1=2	2
2行	0	4	0	0	1	0	16	s_2=16	4
3行	0	0	1	0	0	0.25	3	x_2=3	∞

图3.6 单纯形表1

第四步：继续在第0行找负系数对应的入基变量，发现x_1对应的系数是-2，可以入基。同时比值运算发现s_1对应的变量需要出基，因此第一行、x_1列对应的元素1是转轴点，圈一个圈，并进行行列运算，得图3.7：

	z	x_1	x_2	s_1	s_2	s_3	rhs	Basic Varible	Ratio
0行	1	-2	-3	0	0	0	0	z=0	
1行	0	1	2	1	0	0	8	s_1=8	4
2行	0	4	0	0	1	0	16	s_2=16	∞
3行	0	0	4	0	0	1	12	s_3=12	3
0行	1	-2	0	0	0	0.75	9	z=9	
1行	0	1	0	1	0	-0.5	2	s_1=2	2
2行	0	4	0	0	1	0	16	s_2=16	4
3行	0	0	1	0	0	0.25	3	x_2=3	∞
0行	1	0	0	2	0	-0.25	13	z=13	
1行	0	1	0	1	0	-0.5	2	x_1=2	null
2行	0	0	0	-4	1	2	8	s_2=8	4
3行	0	0	1	0	0	0.25	3	x_3=3	12

图3.7 单纯形表2

第五步：继续上述计算，注意这里因为入基变量s_3对应的列有负值，在比值运算时直接赋值为空，因为比值只看正值，如果将负值也考虑进来取最小比值，同样将导致负的右端项。通过入基变量选取和比值测试，对元素2圈圈，做行列变换，得图3.8：

z	x_1	x_2	s_1	s_2	s_3	rhs	Basic Varible	Ratio
1	-2	-3	0	0	0	0	z=0	
0	1	2	1	0	0	8	s_1=8	4
0	4	0	0	1	0	16	s_2=16	∞
0	0	4	0	0	1	12	s_3=12	3
1	-2	0	0	0	0.75	9	z=9	
0	1	0	1	0	-0.5	2	s_1=2	2
0	4	0	0	1	0	16	s_2=16	4
0	0	1	0	0	0.25	3	x_2=3	∞
1	0	0	2	0	-0.25	13	z=13	
0	1	0	1	0	-0.5	2	x_1=2	null
0	0	0	-4	1	2	8	s_2=8	4
0	0	1	0	0	0.25	3	x_3=3	12
1	0	0	1.5	0.125	0	14	z=14	
0	1	0	0	0.25	0	4	x_1=4	
0	0	0	-2	0.5	1	4	s_3=4	
0	0	1	0.5	-0.125	0	2	x_2=2	

图3.8 单纯形表3

第六步：最新表中发现第0行的所有元素均为正值，此时选取任何变量入基，都会使得z值因为正的缩减成本而降低，很显然这对于最大化问题来说是不利的。因此，上表已经达到最优状态，单纯形法迭代结束。

综上，原问题最优解就是$x_1 = 4, x_2 = 2$,目标函数值$z = 14$.

3.3.3 单纯形法的解

单纯形法在求解过程中，会出现解不单一或者无解的情况，排除求解错误的原因，它们的出现还跟原LP的属性有一定的关系，下面我们针对不同的解的情况进行分析阐述，以下所用例子均来自参考文献[3]。

1. 有唯一最优解的情况

单纯形法求解标准LP问题时，出现唯一最优解的特征是在单纯形表的第0行中，非基变量的系数都是正的（大于0），同时约束右端项全部非负。如上一节我们所用到的例题，最优表如图3.9所示：

z	x_1	x_2	s_1	s_2	s_3	rhs	Basic Varible	Ratio
1	0	0	1.5	0.125	0	14	z=14	
0	1	0	0	0.25	0	4	x_1=4	
0	0	0	-2	0.5	1	4	s_3=4	
0	0	1	0.5	-0.125	0	2	x_2=2	

图3.9 具有唯一最优解特征的最优单纯形表

从表中我们可以看出，基变量为(x_1, x_2, s_3)，非基变量为(s_1, s_2)，非基变量的0行系数全部大于0，右端项全部不小于零，因此该问题有唯一最优解。

让我们来简单理解下其中的原因，第0行变量的系数通常称为缩减成本，指的是相应的变量每增加1（单位）时目标函数所减少的量。在上表中，基变量的缩减成本都为0，非基变量的缩减成本全部为正，也就是说，不论将哪一个非基变量换成基变量，即增加非基变量的取值（非基变量取值在最优解中其实为0），都将使得目标函数减少，这对于最大化问题是不利的，因此当下不能将任何一个非基变量换成基变量，因此已经获得最优解，并且只有此唯一最优解。

2. 有可选最优解的情况

如果一个LP有不止一个最优解，那么我们称之为有可选最优解的LP。由上面的分析，我们很容易推理出来，如果非基变量在最优表中，出现了0系数，即其缩减成本为0，那么它可以作为入基变量继续计算，这时再次得到的单纯形表，目标函数是不变的，也就是说具有相同目标函数的解有不止一个。

3. 无界解的情况

在单纯形表运算中，如果发现非基变量在0行具有负系数，但是该非基变量对应的约束列中，系数也全部为负，这时无法通过比值测试找到可以换出基的变量，比值测试失败（比值测试要选最小的非负数），但是单纯形表还没有到最优状态，这时就是产生了无界解的情况。

无界解的产生多是由于LP的某些变量没有很好地约束住，使得LP在某一个方向上可以任意取值，其中无界方向也是可以求解的，读者若感兴趣可以参看相关研究，这里不作表述。

4. 解的退化和集中

在单纯形法求解中，如果某一步发现某个基变量的值等于0，这时对于某个即将入基的变量，在比值测试时一定会把这个为0的基变量换出，然而新入基的变量值又将为0，这就极有可能出现循环的现象，此时就是解的退化。如图3.10所示，s_2和x_1作为基变量，值均为0，出现了解的退化现象，解的退化会使得单纯形法的求解效率降低，甚至永远无法求出最优解，著名学者Dantzig等人针对这种现象对单纯形法进行了修改，感兴趣的读者可阅读参考文献[4]。

z	x_1	x_2	s_1	s_2	rhs	Basic Variable	Ratio
1	-5	-2	0	0	0	z=0	
0	1	1	1	0	6	s_1=6	6
0	0	-1	0	1	0	s_2=0	0✓
1	0	-7	0	5	0	z=0	
0	0	2	1	-1	6	s_1=6	
0	1	-1	0	1	0	x_1=0	

图3.10 出现退化现象的单纯形表

3.4 变种单纯形法

3.4.1 大M法

用单纯形法求解LP，要求初始单纯形表有基变量，在前面的论述中，我们采用的是加松弛变量来构造初始基，但是如果原LP中有$\geq$约束，那么这个时候不能加松弛变量，构造初始基就变得没那么容易了，这类LP一般可以用大M法来求解，本小节将主要介绍大M法求解LP。

考虑如下LP：

$$
\begin{aligned}
\min \quad & z = 2x_1 + 3x_2 \\
\text{s.t.} \quad & 0.5x_1 + 0.25x_2 \leq 4 \\
& x_1 + 3x_2 \geq 20 \\
& x_1 + x_2 = 10 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

转化为标准形式：

$$
\begin{aligned}
\min \quad & z - 2x_1 - 3x_2 = 0 \\
\text{s.t.} \quad & 0.5x_1 + 0.25x_2 + s_1 = 4 \\
& x_1 + 3x_2 - e_2 = 20 \\
& x_1 + x_2 = 10 \\
& x_1, x_2, s_1, e_2 \geq 0
\end{aligned}
$$

回顾一下使用单纯形法的要求，首先要有基变量组合，同时右端项不能为负。很明显上述问题不满足这个约束，为了顺利地使用单纯形法来求解，我们考虑在第二、三个约束中分别添加一个人工变量，满足构造初始基变量的要求，这个变量是我们人为构造的，不是真正的变量，所以称之为人工变量。

为了保证人工变量不出现在最优解中（人工变量不是真正的变量，出现在最优解中是不被允许的），在最小化问题中，我们可以给人工变量添加一个比较大的系数放入目标函数中，这样由于它的系数比较大，对于目标函数来说是不利的，最优解将不会选择人工变量。新构造的LP如下所示：

$$
\begin{aligned}
\min \quad & z - 2x_1 - 3x_2 - Ma_2 - Ma_3 = 0 \\
\text{s.t.} \quad & 0.5x_1 + 0.25x_2 + s_1 = 4 \\
& x_1 + 3x_2 - e_2 + a_2 = 20 \\
& x_1 + x_2 + a_3 = 10 \\
& x_1, x_2, s_1, e_2, a_2, a_3 \geq 0
\end{aligned}
$$

这样初步构造好了初始基，还需要对目标函数的a_2，a_3两个变量稍做处理（通过行变换消除0行的a_2和a_3），才能使得基变量组合成为真正的基变量，处理之后具有基变量的新的标准型如下所示，之后就可以使用单纯形法进行求解了。

$$
\begin{aligned}
\min \quad & z+(2M-2)x_1+(4M-3)x_2-Me_2=30M \\
\text{s.t.} \quad & 0.5x_1+0.25x_2+s_1=4 \\
& x_1+3x_2-e_2+a_2=20 \\
& x_1+x_2+a_3=10 \\
& x_1,x_2,s_1,e_2,a_2,a_3\geq 0
\end{aligned}
$$

下面用图3.11演示单纯形法求解该LP的具体过程：

z	x_1	x_2	s_1	e_2	a_2	a_3	rhs	Basic Variable	Ratio
1	2M-2	4M-3	0	-M	0	0	30M	z=30M	
0	1/2	1/4	1	0	0	0	4	s_1=4	16
0	1	3	0	-1	1	0	20	a_2=20	20/3✓
0	1	1	0	0	0	1	10	a_3=10	10
1	(2M-3)/3	0	0	(M-3)/3	(3-4M)/3	0	(6+10M)/3	z=(6+10M)/3	
0	5/12	0	1	1/12	-1/12	0	7/3	s_1=7/3	28/5
0	1/3	1	0	-1/3	1/3	0	20/3	x_2=20/3	20
0	2/3	0	0	1/3	-1/3	1	10/3	a_3=10/3	5✓
1	0	0	0	-1/2	(1-2M)/2	(3-2M)/2	25	z=25	
0	0	0	1	-1/8	1/8	-5/8	1/4	s_1=1/4	
0	0	1	0	-1/2	1/2	-1/2	5	x_2=5	
0	1	0	0	1/2	-1/2	3/2	5	x_1=5	

图3.11 单纯形表求解带有M的LP

最小化问题，非基变量系数全部非正，说明已经获得了最优解，至此大M法求解该问题结束。

运用大M法求解时，如果在最优表的基变量中，出现了人工变量，并且其值为正数，这时说明当前LP没有可行解。在实际应用中，大M的取值是一个难题，一般而言，大M至少要比原目标函数中的最大系数大100倍。但这么大的数给计算的四舍五入等操作带来难度，也正是由于这个原因，大多数计算机代码中使用两阶段法来求解上述LP。下面讲述两阶段法求解LP的步骤。

3.4.2 两阶段法

针对初始基不容易得到的LP，除了用大M法来求解，还可以采用两阶段单纯形法。两阶段法在开始阶段的处理上和大M法一样，把人工变量加入到≥约束中，然后在第一阶段中将目标函数设定为最小化所有人工变量之和，然后用单纯形法求解这一问题，第一阶段结束之后，重新引入原LP的目标函数构造第二阶段，并将非基变量中的人工变量删除，进行求解获得最优解。

下面我们仍然将大M法用到的LP来解释两阶段法的具体应用，过程如下：

第一阶段：

构造第一阶段的LP：

$$
\begin{aligned}
\min \quad & w' = a_2 + a_3 \\
\text{s.t.} \quad & 0.5x_1 + 0.25x_2 + s_1 = 4 \\
& x_1 + 3x_2 - e_2 + a_2 = 20 \\
& x_1 + x_2 + a_3 = 10 \\
& x_1, x_2, s_1, e_2, a_2, a_3 \geq 0
\end{aligned}
$$

转化为标准型并进行单纯形求解，如图3.12：

w'	x_1	x_2	s_1	e_2	a_2	a_3	rhs	Basic Variab	Ratio
1	2	4	0	-1	0	0	30	w'=30	
0	1/2	1/4	1	0	0	0	4	s_1=4	16
0	1	3	0	-1	1	0	20	a_2=20	20/3✓
0	1	1	0	0	0	1	10	a_3=10	10
1	2/3	0	0	1/3	-4/3	0	10/3	w'=30	
0	5/12	0	1	1/12	-1/12	0	7/3	s_1=7/3	28/5
0	1/3	1	0	-1/3	1/3	0	20/3	x_2=20/3	20
0	2/3	0	0	1/3	-1/3	1	10/3	a_3=10/3	5✓
1	0	0	0	0	-1	-1	0	w'=0	
0	0	0	1	-1/8	1/8	-5/8	1/4	s_1=1/4	
0	0	1	0	-1/2	1/2	-1/2	5	x_2=5	
0	1	0	0	1/2	-1/2	3/2	5	x_1=5	

图3.12 单纯形表求解第一阶段LP

第一阶段计算结束，人工变量不在最优表的基变量中，这时把最优表中的人工变量删除，然后重新引入原目标函数：$z - 2x_1 - 3x_2 = 0$。这时需要对0行进行基变量构造的处理，否则单纯形表将出现没有基变量的情形，处理的方法是将基变量x_1和在x_2在0行中的系数通过行变换变为0，以使得初始单纯形表中有基变量。所得的第二阶段的初始单纯形表如图3.13所示：

z	x_1	x_2	s_1	e_2	rhs	Basic Variable	Ratio
1	0	0	0	-0.5	25	z=25	
0	0	0	1	-1/8	1/4	s_1=1/4	
0	0	1	0	-1/2	5	x_2=5	
0	1	0	0	1/2	5	x_1=5	

图3.13 单纯形表求解第二阶段LP

单纯形图3.13的非基变量0行系数为负，最小化问题来说已经获得了最优解，因此该单纯形表已经是最优单纯形表，最优解已经获得，与大M法获得的解相同。

需要注意的是，运用两阶段法求解LP时，第一阶段的结果并不一定如上述过程一样这么幸运（求解出的人工变量全都不在最优基中），其结果可能有三种情况：

第一种，正如上面例子所示，第一阶段目标函数w'的最优值为0，并且人工变量都不在第一阶段最优基中，这时第二阶段的操作比较简单，直接消除人工变量，并将原始目标函数引入构造第二阶段单纯形表进行计算即可，第二阶段的最优解就是原始问题的最优解。

第二种，第一阶段目标函数w'的最优值为0，但至少有一个人工变量在第一阶段最优基中（其值为0），这时在构造第二阶段单纯形表时，不仅要从第一阶段最优表中删除所有非基人工变量列，还要删除0行中具有负系数的原问题的变量，即便这个变量在原始问题的目标函数中也要将其删除，然后再用构造初始基的行变换方法构造第二阶段单纯形表，第二阶段单纯形表的解就是原始LP的解。

第三种，第一阶段目标函数w'的值大于0，这时说明人工变量存在于第一阶段的最优表基变量中并且不为0，这说明原LP没有解。

至此，单纯形法求解LP问题的主要方法介绍完毕，无论是大M法还是两阶段法，都是基于原始单纯形法的理论基础，先找出一个基本可行解，然后判断是否最优，若不是最优，则按照入基、出基规则进行出入基的变换，直到获得最优解。

本章参考文献

[1]胡运权，郭耀煌，《运筹学教程（第三版）》，清华大学出版社，P19—20.

[2] Dimitris Bertsimas, John N. Tsitsiklis, Introduction to Linear Optimization. Athena Scientific, Belmont, Massachusetts. P120—123.

[3] Operations Research Applications and Algorithms (Forth Edition). Wayne L., Winston. Thomson Learning. P166—P200.

[4] Robert G. Bland，New Finite Pivoting Rules for the Simplex Method，1977

第4章 对偶理论和敏感度分析

作者：翁欣，清华大学博士在读

研究方向：供应链优化，服务运营，随机优化

这一章，我们将一线性规划问题设为原问题（primal problem），引入另一线性规划，称为该问题的对偶问题（dual problem），将介绍对偶问题的写法和对偶理论。对偶理论引出了一个新的求解算法——对偶单纯形法。最后我们引入了敏感度分析方法去探讨数据变化对求解结果的影响。

4.1 对偶问题

4.1.1 对偶问题的理解

线性规划里，每个原始线性规划问题，即原问题（**primal problem**）都有相应的另一个线性规划问题——对偶问题（**dual problem**），而对偶的对偶又是原问题本身。既然我们已经知道用单纯形法去求解原问题了，为什么还要了解它的对偶问题呢？哪些情况下，考虑对偶问题有助于求解原问题？下面我们从两个角度阐述。

1.经济学角度

这是一个在教材上被广泛使用的解释：如果原问题是企业A拥有m种资源（有m个约束），计划生产n种产品（有n个变量），目标是最大化总收入；那么对偶问题就是，企业B想要收购这些资源，需要确定m种资源的报价（有m个变量），目标是最小化总成本，但企业A只有在卖资源的收益不低于卖产品的时候才会同意卖资源（n个约束）。

我们通过一个数值实例来进一步解释：

企业A生产书架、桌子和椅子三种产品，拥有48m^3木材，20h制造工时，8h测试工时三项资源。已知：书架售价60元，生产一个书架需要8单位木材、4单位制造工时和2单位测试工时；桌子售价30元，生产一张桌子需要6单位木材、2单位制造工时和1.5单位测试工时；椅子售价20元，生产一把椅子需要1单位木材、1.5单位制造工时和0.5单位测试工时；求三种产品各生产多少数量时，企业A能够最大化总收入。

原问题的线性规划为：

$$\max \quad 60x_1 + 30x_2 + 20x_3 \text{（卖出三种产品的总收入）}$$

$$\text{s.t.} \quad 8x_1 + 6x_2 + x_3 \leq 48 \text{（木材总量约束）}$$

$$4x_1 + 2x_2 + 1.5x_3 \leq 20 \text{（制造工时总量约束）}$$

$$2x_1 + 1.5x_2 + 0.5x_3 \leq 8 \text{（测试工时总量约束）}$$

$$x_1, x_2, x_3 \geq 0$$

如果企业B想要收购木材、制造工时、测试工时这三样资源，就必须为每项资源报价，并且满足企业A愿意出让资源的条件，即让企业A获得不低于自己制造产品的收入。求三项资源的单位报价各为多少时，企业B能够最小化总成本。

那么，可以写出对偶问题的线性规划是：

$$\min \quad 48y_1 + 20y_2 + 8y_3 \text{（收购三种资源的总成本）}$$

$$\text{s.t.} \quad 8y_1 + 4y_2 + 2y_3 \geq 60 \text{（把足够制造一个书架的资源卖掉，获得的收入必须不低于书架售价）}$$

$$6y_1 + 2y_2 + 1.5y_3 \geq 30 \text{（把足够制造一张桌子的资源卖掉，获得的收入必须不低于桌子售价}$$

$$y_1 + 1.5y_2 + 0.5y_3 \geq 20 \text{（把足够制造一把椅子的资源卖掉，获得的收入必须不低于椅子售价}$$

$$y_1, y_2, y_3 \geq 0 \text{（分别是一单位木材、制造工时和测试工时的报价）}$$

上面的例子，旨在从实际经济问题的角度解释「每一个线性规划问题都存在一个与其对偶的问题」这句话的合理性。

2.数学角度

从数学角度，对偶问题可以被理解为寻找原问题目标函数上界（或下界）的问题。

以原问题是求目标函数最大化为例，我们以向量形式写线性规划：

$$\max \quad z = \mathbf{cx}$$

$$\text{s.t.} \quad A\mathbf{x} \leq \mathbf{b}$$

$$\mathbf{x} \geq 0$$

其中$\mathbf{c}$为$1 \times n$向量，$\mathbf{x}$为$n \times 1$向量，A为$m \times n$矩阵，$\mathbf{b}$为$m \times 1$向量。

对于任意一个非负的$n \times 1$向量$\mathbf{y} \geq 0$和可行解$\mathbf{x}$，原问题约束两边同乘$\mathbf{y}^T$，有：

$$\mathbf{y}^T A\mathbf{x} \leq \mathbf{y}^T\mathbf{b}$$

如果找到一个$\mathbf{y}^T$满足$\mathbf{c} \leq \mathbf{y}^T A$，那么对所有可行解$\mathbf{x}$，都有：

$$\mathbf{c}\mathbf{x} \leq \mathbf{y}^T A\mathbf{x} \leq \mathbf{y}^T\mathbf{b}$$

这意味着，$\mathbf{y}^T\mathbf{b}$就是原问题的一个上界。

换言之，所有满足$\mathbf{c} \leq \mathbf{y}^T A$的$\mathbf{y}$对应的$\mathbf{y}^T\mathbf{b}$，都是原问题的上界。

那么，最小的上界就是原问题目标函数的最优值。在所有的上界中，我们要找到最小的那一个，这个问题表述出来就是：

$$\min \quad w = \mathbf{b}^T\mathbf{y}$$

$$\text{s.t.} \quad A^T\mathbf{y} \geq \mathbf{c}^T$$

$$\mathbf{y} \geq 0$$

可以发现，这就是对偶问题的形式。反之，如果原问题是求目标函数最小化，那么对偶问题就是在寻找原问题目标函数的下界。图4.1可以帮助理解：26是目标函数最优值，两个问题分别从左右两侧逼近最优值。

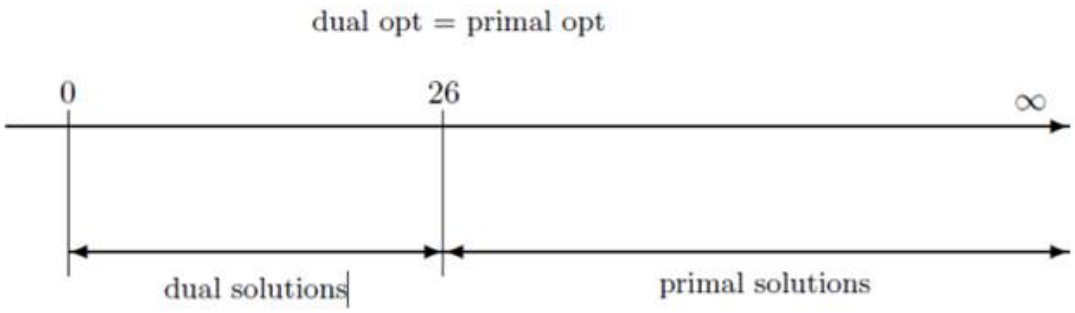


图4.1 原问题和对偶问题从两侧逼近最优解

3.结合两个角度的解释

依然从刚才的例子出发，现在有一组生产方案满足原问题约束，对应企业A总收入是$\bar{z}$；有一组报价满足对偶问题约束，对应企业B总成本是$\bar{w}$。

一定有$\bar{z} \leq \bar{w}$，也就是企业B付出的总成本一定不低于企业A的总收入（这是对偶问题的约束，所以一定成立）。

也就是说，在这个例子里，对偶问题的任意一个可行解，都是原问题的一个上界。因为企业B用这组报价可以买到资源，所以企业A制造产品的收入不可能高于这组报价对应的成本。

当我们找到企业B所有报价里总成本最低的那一组，就是企业A能获得的总收入的最大值，是原问题的最优解。

4.1.2 对偶问题的优势

接下来我们考虑，同样是求解线性规划，解对偶问题就会比解原问题更容易吗？或者说，什么情况下，考虑对偶问题有助于求解原问题？

1. 原问题约束多、变量少时，求解对偶问题能够降低计算时间。

使用单纯形法时，如果原问题约束多变量少，转换成对偶问题，就是约束少变量多。根据单纯形法的原理，约束的减少能够有效降低计算时间。

2. 帮助证明原问题无解。

类似“证明无罪比证明有罪更难”，要证明原问题有解，只需要找出一个满足约束的点，却不能通过遍历所有的点来证明原问题无解。对偶问题的出现为证明原问题无解提供了思路。

3. 便于进行敏感度分析。

很多时候我们对原问题的好奇心并不仅限于得到最优解，而是还关注「如果某些已知条件发生变化，对最优解的影响程度如何」，这就是敏感度分析。对偶问题和敏感度分析息息相关：一是增加敏感度分析的直观程度（例如，对偶问题的最优解就是原问题约束的影子价格），二是在改变某些条件导致原问题无可行解时，可以借助仍然有可行解的对偶问题来分析。

4.1.3 对偶问题的表达

上两小节我们解释了「为何引入对偶问题」，这一节将阐述对偶问题的表达，即已知原问题时，如何快速写出对偶问题。先介绍一般形式的对偶问题写法，再介绍如何将非一般形式转化为一般形式，最后总结。

1. 一般形式的对偶问题

以目标函数最大化问题为例，原问题的一般形式指的是变量全部非负、约束全部为小于等于约束的线性规划，向量形式如下：

$$\max \quad z = \mathbf{cx}$$

$$\text{s.t.} \quad A\mathbf{x} \leq \mathbf{b}$$

$$\mathbf{x} \geq 0$$

根据上一小节内容，其对偶问题的含义是：在原问题目标函数的所有上界中，找到最小的一个。相应对偶问题的一般形式，指的是变量全部非负、约束全部为大于等于约束的线性规划，向量形式如下：

$$\min \quad w = \mathbf{b}^T\mathbf{y}$$

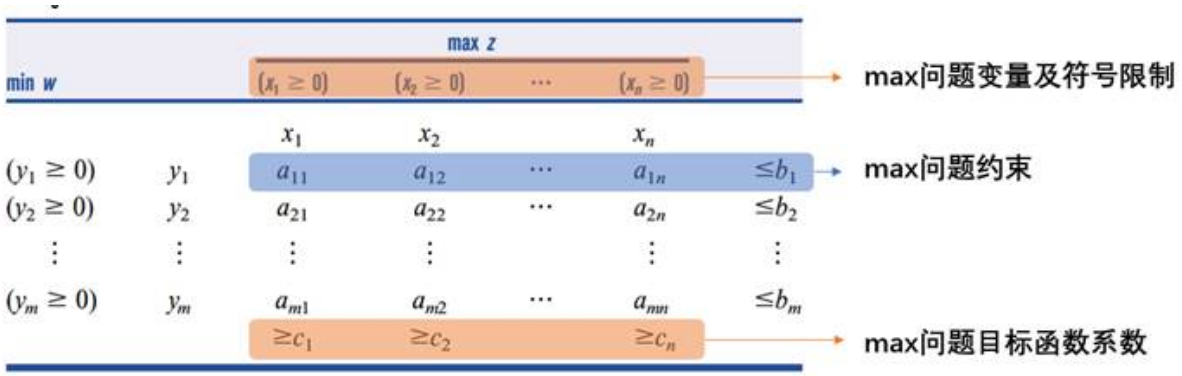
$$\text{s.t.} \quad A^T\mathbf{y} \geq \mathbf{c}^T$$

$$\mathbf{y} \geq 0$$

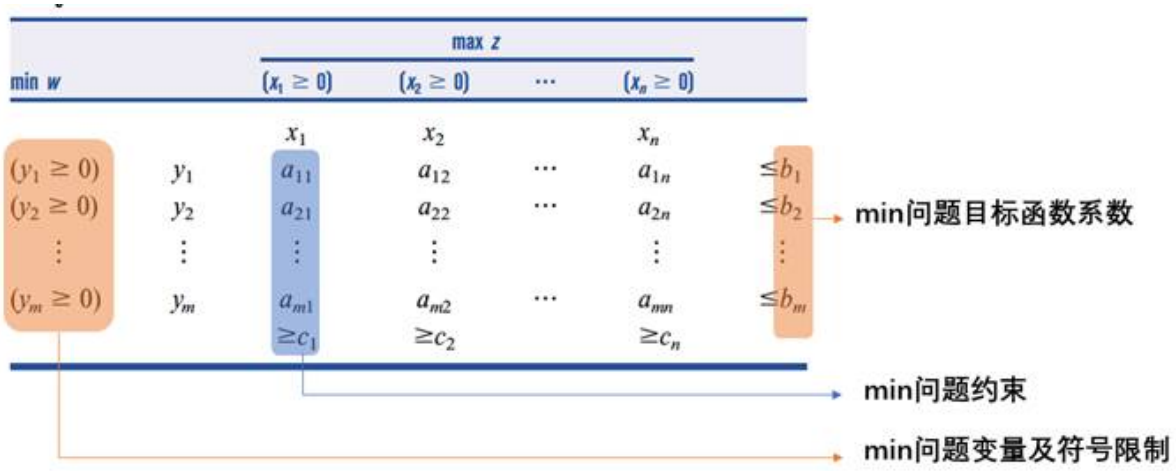
在向量形式中，可以直观看出原问题和对偶问题的系数存在对应关系，例如：原问题目标函数的系数向量的转置$\mathbf{c}^T$，是对偶问题约束的右侧向量，原问题中约束条件中的向量$\mathbf{b}$则出现在对偶问题的目标函数中；原问题约束的系数矩阵的转置A^T，是对偶问题约束的系数矩阵。借用上一节的实例，对应关系如图4.2：

图4.2 原问题与对偶问题对应关系

如果我们面对的是一般形式问题，可以使用表格法便捷地写出对偶问题。图4.3中，由上至下按行读是原问题的约束和目标函数，由左至右按列读是对偶问题的约束和目标函数。



(a) 标示max问题



(b) 标示min问题

图4.3 表格法写对偶问题

2. 非一般形式的对偶问题

实际中我们常常会遇到非一般形式的线性规划问题，在应用表格法之前需要先对变量和约束的形式进行转化（以目标函数最大化问题为例）：

①如果有≥约束，则左右同乘-1，转化为≤约束；

②如果有=约束，则先转换为一个≤约束和一个≥约束，再按①把≥约束转换为≤约束。例如，如果存在约束$x_1 + x_2 = 2$，等价于同时满足$x_1 + x_2 \leq 2$和$x_1 + x_2 \geq 2$，后者再写成$-x_1 - x_2 \leq -2$。

③如果有无符号限制(unrestricted in sign, **urs**)的变量，用两个符号限制为≥ 0的变量相减表示，转换关系为：$x(x \quad \mathbf{urs}) = x' - x''(x', x'' \geq 0)$。

3. 总结

将原问题转换为对偶问题的通用步骤归纳如下：①将原问题转换为标准形式；②由上至下逐行写原问题一般形式的变量及符号限制、约束和目标函数；③根据对应关系表确定对偶问题的约束符号和变量符号；④由左至右逐列读，得到对偶问题的变量及符号限制、约束和目标函数。

表格法中，原问题和对偶问题的符号对应关系可参考表4.1。（由于对偶的对偶是原问题，所以当原问题是最小化问题时，对应关系同样适用）

表4.1 原问题与对偶问题对应关系

原问题			对偶问题
变量向量	$\mathbf{x}(n \times 1)$	$\mathbf{y}(m \times 1)$	变量向量
目标函数方向	max	min	目标函数方向

原问题			对偶问题
目标函数系数向量	$\mathbf{c}(1\times n)$	$\mathbf{b}^T(1\times m)$	目标函数系数向量
约束系数矩阵	$A(m\times n)$	$A^T(n\times m)$	约束系数矩阵
约束右侧常数向量	$\mathbf{b}(m\times 1)$	$\mathbf{c}^T(n\times 1)$	约束右侧常数向量
变量符号	≥ 0	$\geq c_i$	约束符号
	≤ 0	$\leq c_i$	
	urs	$= c_i$	
约束符号	$\leq b_i$	≥ 0	变量符号
	$\geq b_i$	≤ 0	
	$= b_i$	urs	

本章到目前已经回答了「为什么要引入对偶问题」以及「如何写对偶问题」，下一节的主题是对偶原理，回答的问题是「为什么原问题和对偶问题具有相同的最优解（在有最优解的前提下）」，在证明的过程中，也会发现线性规划的一些其他性质。

4.2 对偶原理

这一节着重讲述原问题和对偶问题的解之间的关系，即对偶原理。通过对偶原理，我们可以更好地理解如何借助对偶问题的解获得原问题的解。本节先概述弱对偶定理、强对偶定理和互补松弛定理的内容，再逐个进行说明。

以原问题是最大化问题为例，弱对偶定理是与可行解相关的一个定理：当对偶问题有任一可行解$\mathbf{y}_0$时，其相应的对偶问题目标函数值w_0是原问题目标函数值的上界。换句话说，对原问题任一可行解$\mathbf{x}_0$对应的原问题目标函数z_0，一定有$z_0 \leq w_0$成立。在4.1.1节解释为什么每一个线性规划都一定存在对偶问题时，我们已经简单说明过这一定理，接下来将在4.2.1节给出更详细的证明。

强对偶定理则与最优解相关：当已知原问题最优解时，对偶问题也一定有最优解，且两个问题最优解的目标函数值相等。

当原问题和对偶问题各自有一个可行解$\mathbf{x}_0$和$\mathbf{y}_0$时，要判断$\mathbf{x}_0$和$\mathbf{y}_0$是否为最优解，就涉及互补松弛定理：$\mathbf{x}_0$和$\mathbf{y}_0$分别为原问题和对偶问题最优解的充要条件是，

$$\mathbf{y}_0\mathbf{s} = \mathbf{y}_0(A\mathbf{x}_0 - \mathbf{b}) = 0$$

$$\mathbf{x}_0^T\mathbf{e} = \mathbf{x}_0^T(A^T\mathbf{y}_0 - \mathbf{c}^T) = 0$$

其中$\mathbf{s}$和$\mathbf{e}$分别为原问题和对偶问题的松弛变量(slack variables)和剩余变量(excess variables)。

借助这些定理，给出对偶问题的最优解，便可求得原问题的最优解；给出对偶问题的可行解，也可限定原问题最优解的范围，反之亦然。接下来对每个定理进行详细说明。

4.2.1 弱对偶定理

以原问题是最大化问题为例，原问题和对偶问题的一般形式见4.1.3节。

已知原问题和对偶问题各自有一个可行解$\mathbf{x}_0$和$\mathbf{y}_0$。由于$\mathbf{y}_0$为非负向量，因此在原问题约束两边同乘$\mathbf{y}_0$，得到以下公式，对任意$\mathbf{x}$都成立：

$$\mathbf{y}_0^T A\mathbf{x} \leq \mathbf{y}_0^T \mathbf{b}$$

同理，在对偶问题约束两边同乘$\mathbf{x}_0$，得到以下公式，对任意$\mathbf{y}$都成立：

$$\mathbf{x}_0^T A^T \mathbf{y} \geq \mathbf{x}_0^T \mathbf{c}^T \Longleftrightarrow \mathbf{y}^T A\mathbf{x}_0 \geq \mathbf{c}\mathbf{x}_0$$

对以上两个公式，分别代入$\mathbf{x}_0$和$\mathbf{y}_0$后合并，得到：

$$\mathbf{c}\mathbf{x}_0 \leq \mathbf{y}_0^T A\mathbf{x}_0 \leq \mathbf{y}_0^T \mathbf{b} \Longleftrightarrow z_0 \leq w_0$$

即原问题目标函数z_0一定不高于对偶问题目标函数w_0，弱对偶定理得以证明。

基于弱对偶定理，还可以得到两个推论：

1. （最优性）如果原问题和对偶问题各自有一个可行解$\mathbf{x}_0$和$\mathbf{y}_0$，且相应的目标函数值$z_0 = w_0$，那么$\mathbf{x}_0$和$\mathbf{y}_0$分别是原问题和对偶问题的最优解。
2. 如果原问题的解无界(unbounded)，那么对偶问题无可行解；如果原问题无可行解，那么对偶问题的解无界。

4.2.3 强对偶定理

强对偶定理的完整表述是：如果BV是原问题的一组最优基，那么$\mathbf{c}_{BV}B^{-1}$是对偶问题的一组最优解，$\mathbf{c}_{BV}B^{-1}\mathbf{b}$是原问题和对偶问题最优解对应的目标函数值。

其证明思路分为三步：

1. 已知BV是原问题的一组最优基，证明$\mathbf{c}_{BV}B^{-1}$是对偶问题的一组可行解。此时原问题和对偶问题各有一组可行解。
2. 证明$\mathbf{c}_{BV}B^{-1}$对应的对偶问题目标函数值等于BV对应的原问题目标函数值，即两组可行解的目标函数值相同。
3. 前两步的证明结果恰好是弱对偶定理最优性推论的条件，因此可以推出这两组可行解分别是原问题和对偶问题的最优解。

4.2.3 互补松弛定理

互补松弛定理是关联原问题和对偶问题解的重要定理，在证明之前，首先定义原问题的松弛变量$\mathbf{s}$和对偶问题的剩余变量$\mathbf{e}$。

已知$\mathbf{x}_0$是原问题的一个可行解，松弛变量$\mathbf{s}$描述的是$\mathbf{x}_0$代入原问题时约束的“松弛”程度。将约束$A\mathbf{x} \leq \mathbf{b}$右边的常量$\mathbf{b}$理解为资源，当约束是以等号成立时，说明此时资源正好全部用完，没有富余；当约束时以不等号成立时，说明此时资源还没有被全部利用，存在富余，有一定的“松弛”。同理，剩余变量$\mathbf{e}$描述的是对偶问题约束$A^T\mathbf{y} \geq \mathbf{c}^T$的“剩余”程度，两者的计算公式如下：

$$\mathbf{s} = A\mathbf{x}_0 - \mathbf{b}$$

$$\mathbf{e} = A^T\mathbf{y}_0 - \mathbf{c}^T$$

互补松弛定理的内容是，可行解$\mathbf{x}_0$和$\mathbf{y}_0$分别为原问题和对偶问题最优解的充要条件是

$$\mathbf{y}_0\mathbf{s} = 0$$

$$\mathbf{x}_0^T\mathbf{e} = 0$$

第一个公式有两种成立情况，分别可以用文字表述为：

- 原问题第i个松弛变量大于0，此时对偶问题第i个变量等于0；
- 对偶问题第i个变量大于0，此时原问题第i个松弛变量等于0；

第二个公式同样有两种成立情况，用文字表述为：

- 对偶问题第j个剩余变量大于0，此时原问题第j个变量等于0；
- 原问题第j个变量大于0，此时对偶问题第j个剩余变量等于0。

上述四种情况意味着，如果原问题/对偶问题中任一个约束时是以非等号形式成立的（nonbinding，不具约束力，即$\mathbf{s}_i$或$\mathbf{e}_j$大于0），那么在对偶问题/原问题中该约束对应的变量就一定为**0**，因此这一定理称为互补松弛。通过互补松弛定理，给出原问题的最优解，便可求得其对偶问题的最优解，反之亦然。

本节介绍了弱对偶定理、强对偶定理和互补松弛定理，阐述了原问题和对偶问题的解之间的关系。下一节「对偶单纯形法」将进一步利用这些关系，介绍利用对偶可行性逐步搜索出原始问题最优解的方法。

4.3 对偶单纯形法

第三章介绍了求解线性规划问题的经典算法——单纯形法，这一节介绍另一种求解算法——对偶单纯形法。这一方法基于对偶原理，利用对偶可行性逐步搜索出原始问题最优解。本节先概述对偶单纯形法的原理、优点，再介绍具体求解步骤和一个算例。

4.3.1 基本思想

在4.1节中我们提到过，原问题和对偶问题是分别从左右两侧逼近最优值。以图4.1为例，左侧是对偶问题的可行解，右侧是原问题的可行解，当两个可行解有相同的目标函数值时（图中的26），这两个解分别是原问题和对偶问题的最优解。

单纯形法是先找到原问题可行解，再通过努力实现对偶可行性（最优性检验）找到最优解。对偶单纯形法可以看作是单纯形法的镜像，先找到对偶问题可行解，再通过努力实现原问题可行来靠近最优解。也可以理解为，我们是把单纯形法应用在对偶问题。

对偶单纯形法在某些特殊情况下比单纯形法更方便：

1. 需要引入多个人工变量来构造原问题可行解时，找到对偶问题可行解可能更容易一些，找到最优解的迭代次数也可能更少。
2. 对偶单纯形法可以与敏感度分析结合使用。假设用单纯形法得到了一个最优解，但需要对数据做一些小修改。在新的条件下，如果原来的最优解不再是原问题可行解，但仍然满足最优性检验，则可以从对偶可行解开始应用对偶单纯形法，更快找到新问题最优解。
3. 在求解某些大型线性规划问题时，对偶单纯形法也有一定的实用价值。CPLEX计算经验表明，对偶单纯形法在解决实际中遇到的大型线性规划问题时通常比单纯形方法更有效。

4.3.2 求解步骤

对偶单纯形法的规则与单纯形法的规则非常相似，区别就是选择进入变量(entering variable)和离开变量(leaving variable)的标准以及停止算法的标准不同。

以目标函数最大化问题为例，具体求解步骤如下。

步骤**1** 初始化

所有约束转化为$\leq$形式；找到一个基本解，使得目标函数中基本变量的系数为0，非基本变量的系数非负（这样如果这个解是可行的，则可保证它是最优的）。

步骤**2** 可行检验

检验是否所有的基本变量都为非负。如果是，那么这一解是可行解，也是最优解，算法结束。否则，进行迭代。

步骤**3** 迭代

1）确定离开变量：在值为负的基本变量中，选择绝对值最大的。

2）确定进入变量：在包含离开变量的约束中选择系数为负的非基本变量，进而在其中选择（目标函数系数/在该约束中的系数）比值绝对值最小的。

3）确定一个新的基本解，返回步骤2检验。

例题**4.1** 用对偶单纯形法解下列问题：

$$\begin{aligned} \max \quad & z = -4y_1 - 12y_2 - 18y_3 \\ \text{s.t.} \quad & -y_1 - 3y_3 + y_4 = -3 \\ & -2y_2 - 2y_3 + y_5 = -5 \\ & y_1, y_2, y_3, y_4, y_5 \geq 0 \end{aligned}$$

其中y_4和y_5为松弛变量。

步骤**1** 初始化

首先找到一组基本解，$BV = \{y_4, y_5\}$，$NBV = \{y_1, y_2, y_3\}$。令NBV中的变量为0，这一组基本解对应的值是：$y_1 = 0, y_2 = 0, y_3 = 0, y_4 = -3, y_5 = -5, Z = 0$。

步骤**2** 可行检验

由于基本变量存在负值，因此不是可行解，需要确定离开变量和进入变量，寻找新一组基本解。

步骤**3** 迭代

（1）从BV中确定离开变量：基本变量中y_4和y_5都为负值，其中y_5的绝对值更大，因此选择y_5作为离开变量；

（2）从NBV中确定进入变量：y_5所在的约束中系数为负的非基本变量是y_2和y_3，两者的（目标函数系数/在该约束中的系数）比值绝对值分别是|12/-2|和|18/-2|，前者更小，所以选择y_2作为进入变量。

（3） 由此产生新一组基本解：$BV = \{y_4, y_2\}$，$NBV = \{y_1, y_5, y_3\}$。返回步骤2进行可行检验。完整求解过程见图4.4，经过两次迭代后找到最优解。

Iteration	Basic Variable	Eq.	Coefficient of:						Right Side
			Z	y_1	y_2	y_3	y_4	y_5	
0	Z	(0)	1	4	12	18	0	0	0
	y_4	(1)	0	−1	0	−3	1	0	−3
	y_5	(2)	0	0	−2	−2	0	1	−5
1	Z	(0)	1	4	0	6	0	6	−30
	y_4	(1)	0	−1	0	−3	1	0	−3
	y_2	(2)	0	0	1	1	0	$-\frac{1}{2}$	$\frac{5}{2}$
2	Z	(0)	1	2	0	0	2	6	−36
	y_3	(1)	0	$\frac{1}{3}$	0	1	$-\frac{1}{3}$	0	1
	y_2	(2)	0	$-\frac{1}{3}$	1	0	$\frac{1}{3}$	$-\frac{1}{2}$	$\frac{3}{2}$

图4.4 算例迭代过程

本节介绍了由对偶理论引出的一个新求解算法——对偶单纯形法的原理、优点以及求解步骤，并指出对偶单纯形法的优点之一是便于与敏感度分析结合使用，下一节「敏感度分析」将具体探讨数据变化对求解结果的影响。

4.4 敏感度分析

除了获得线性规划最优解以外，很多时候我们也关心参数变动对最优解的影响，或者说最优解对参数的依赖情况，这就是「敏感度分析」的研究内容。本节先用一个简单例子直观理解敏感度分析的含义，再讨论分析公式以及三类参数变化情况。

4.4.1 直观图解

考虑一个玩具制造问题。x_1和x_2分别表示每周soldiers和trains两种玩具的制造数量，两种玩具的利润分别是3元和2元，目标函数是最大化利润，有Finishing、Carpentry和Demand三项资源约束。线性规划如下：

$$
\begin{aligned}
\max \quad & z = 3x_1 + 2x_2 \\
\text{s.t.} \quad & 2x_1 + x_2 \leq 100 \quad (Finishing \quad constraint) \\
& x_1 + x_2 \leq 80 \quad (Carpentry \quad constraint) \\
& x_1 \leq 40 \quad (Demand \quad constraint) \\
& x_1, x_2 \geq 0
\end{aligned}
$$

上述线性规划的最优解是$x_1 = 20, x_2 = 60, z = 180$，即生产20个soldiers和60个trains，总收入可以达到最高值180。

由于这一线性规划只有两个变量，我们可以在二维平面上绘制。如图4.5，三条实线分别表示三个约束，和坐标轴一起组成了可行域（黄色区域），虚线为目标函数的等利润线（两种玩具的售价已知，因此斜率固定为-3/2，平移改变z的值）。当平移到B点时，z取最大值，因此可行域中的最优解在B点。

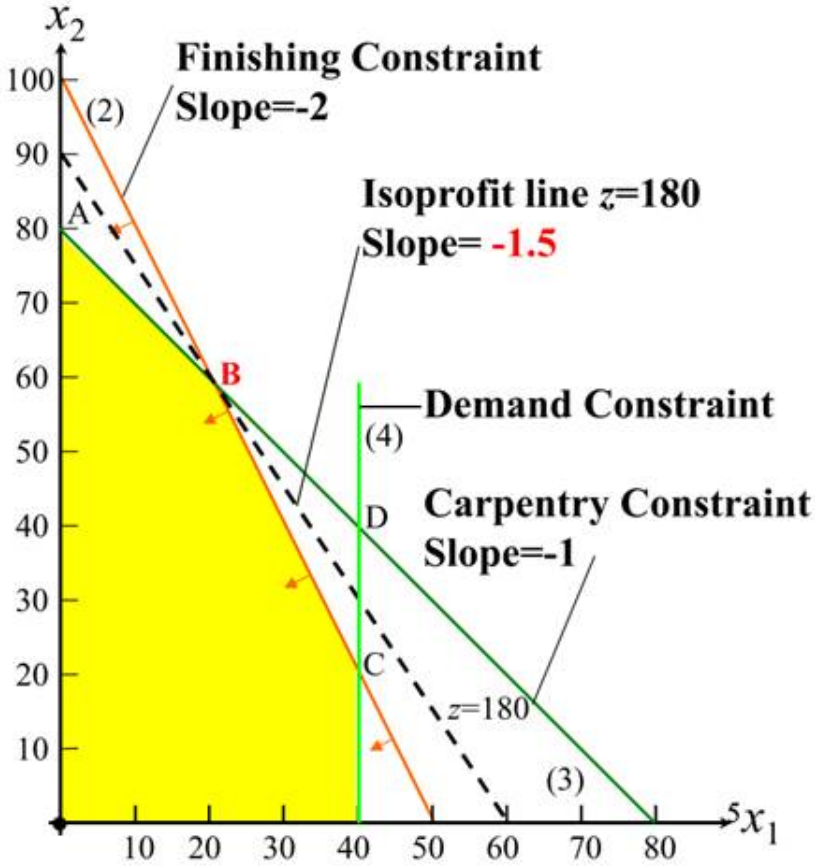


图4.5 最优解示意图（斜率=-1.5）

此时，如果soldiers的利润发生变化，从3元变更为另一个常数c_1，目标函数变更为$max \quad z = c_1x_1 + 2x_2$，等利润线的斜率为$-c_1/2$。可以发现，当价格变化时，等利润线的斜率变化，最优解的位置也随之变化。例如$c_1 = 1$时，soldiers的利润为1元，等利润线的斜率为-0.5，此时在可行域中A点取到最高总利润，如图4.6虚线所示：

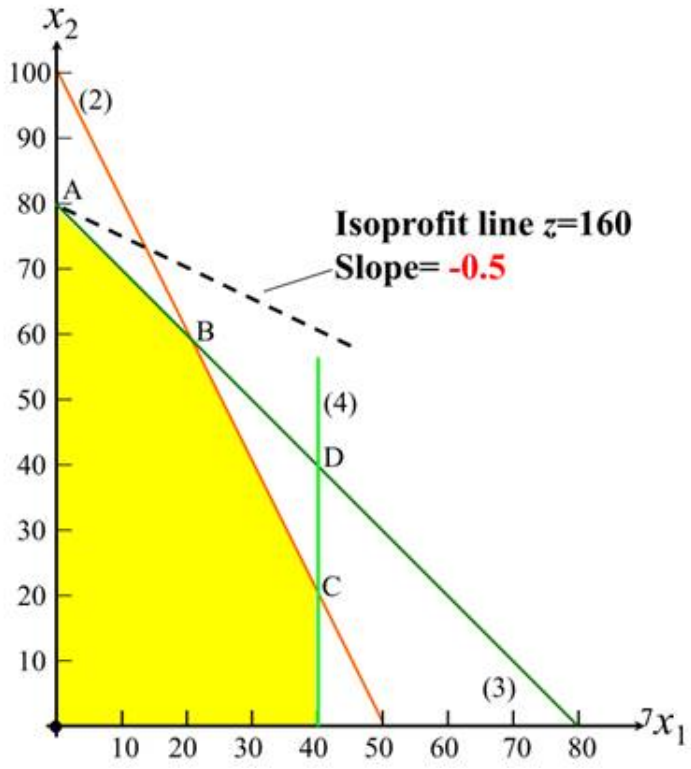


图4.6 最优解示意图（斜率=-0.5）

借助斜率稍作计算，可以得到c_1变化对最优解的影响，即针对c_1的敏感度分析：

- 当soldiers的利润低于2元时，等利润线比约束(3)更平缓，最优解在点A；
- 当soldiers的利润在2元到4元之间时，等利润线的斜率介于约束(2)和约束(3)之间，最优解在点B；
- 当soldiers的利润高于4元时，等利润线比约束(2)更陡，最优解在点C。

基于上述分析，当soldiers的利润发生变动时，我们可以迅速判断变动是否影响现有生产方案，而不必重新求解线性规划。这就是敏感度分析的重要价值之一（尤其是在大规模问题中）：参数发生变化时，可以根据原问题直接计算出最优解的变化情况（原最优解是否仍然可行？如果可行，是否仍是最优？），而不需要重新求解线性规划。

4.4.2 分析公式

当问题不止两个变量时，我们无法在平面图中根据斜率分析，因此需要寻找更普适的方法，即原最优解在新问题中的可行性条件和最优性条件。

首先回顾标准形式线性规划问题：

$$\max \quad z = \mathbf{cx}$$

$$\text{s.t.} \quad A\mathbf{x} \leq \mathbf{b}$$

$$\mathbf{x} \geq 0$$

按照基本变量和非基本变量，我们把原问题中的变量和系数拆分为两部分，拆分后的表述如下表所示。

表4.2 线性规划中的变量和系数

	总称	基本变量部分	非基本变量部分
变量	$\mathbf{x}:n\times 1$	$\mathbf{x}_{BV}:m\times 1$	$\mathbf{x}_{NBV}:(n-m)\times 1$
目标函数系数	$\mathbf{c}:1\times n$	$\mathbf{c}_{BV}:1\times m$	$\mathbf{c}_{NBV}:1\times(n-m)$
约束系数	$A:m\times n$ $\mathbf{a}_j:m\times 1$	$B:m\times n$	$N:m\times(n-m)$

其中，$\mathbf{a}_j:m\times 1$特指变量$\mathbf{x}_j$的约束系数向量。

由此，我们可以把线性规划改写为：

$$\begin{aligned}\max\quad & z=\boldsymbol{c}_{BV}\boldsymbol{x}_{BV}+\boldsymbol{c}_{NBV}\boldsymbol{x}_{NBV}\\ \text{s.t.}\quad & B\boldsymbol{x}_{BV}+N\boldsymbol{x}_{NBV}=\boldsymbol{b}\\ & \boldsymbol{x}_{BV},\boldsymbol{x}_{NBV}\geq 0\end{aligned}$$

约束左乘B^{-1}，得到：

$$B^{-1}B\mathbf{x}_{BV}+B^{-1}N\mathbf{x}_{NBV}=B^{-1}\mathbf{b}$$

$$\Longleftrightarrow \mathbf{x}_{BV}=B^{-1}\mathbf{b}-B^{-1}N\mathbf{x}_{NBV}$$

用这一等式替代目标函数中的$\mathbf{x}_{BV}$，则目标函数中只包含参数和非基本变量，变更为如下形式：

$$\begin{aligned}\max\quad & z=\mathbf{c}_{BV}(B^{-1}\mathbf{b}-B^{-1}N\mathbf{x}_{NBV})+\mathbf{c}_{NBV}\mathbf{x}_{NBV}\\ & =\mathbf{c}_{BV}B^{-1}\mathbf{b}-(\mathbf{c}_{BV}B^{-1}N-\mathbf{c}_{NBV})\mathbf{x}_{NBV}\\ \text{s.t.}\quad & \mathbf{x}_{BV}+B^{-1}N\mathbf{x}_{NBV}=B^{-1}\mathbf{b}\\ & \mathbf{x}_{BV},\mathbf{x}_{NBV}\geq 0\end{aligned}$$

上述线性规划中的参数如下：

- 变量$\mathbf{x}_j$的约束系数向量为$B^{-1}\mathbf{a}_j$；
- 约束右端常量为$B^{-1}\mathbf{b}$；

- 非基本变量$\mathbf{x}_j$的reduced cost（即目标函数系数的负值）为$\mathbf{c}_{BV}B^{-1}\mathbf{a}_j - c_j$，用$\bar{c_j}$表示；
- 目标函数右端常量为$\mathbf{c}_{BV}B^{-1}\mathbf{b}$。

根据单纯形法的最优解求解规则（最大化问题）：基于一组基本变量BV的解是最优解，当且仅当每个约束右端常量非负，并且每个变量的目标函数系数非正。我们推出两个条件。

可行性条件：$B^{-1}\mathbf{b} \geq 0$。当一组BV能使得每个约束右端常量非负的时候，这组BV对应的解可行。

最优性条件：$\bar{c}_j = \mathbf{c}_{BV}B^{-1}\mathbf{a}_j - c_j \geq 0, \forall j \in NBV$。只要每个非基本变量的reduced cost都大于0，即不可能再通过增加这一变量来提高总收入，这组BV对应的解是最优解。

通过分析参数变动是否影响两个条件的结果，我们可以判断原最优解是否仍然可行/最优。接下来，我们详细阐述三类参数变化对最优解的影响。

4.4.3三类参数变化

三类参数变化指的是：目标函数系数$\mathbf{c}$变化、约束右端常量$\mathbf{b}$变化和约束系数A变化。

1. 目标函数系数变化

包含两种情况：非基本变量的目标函数系数$\mathbf{c}_{NBV}$变化，或基本变量的目标函数系数$\mathbf{c}_{BV}$变化。不论哪一种，原最优解一定仍然可行，但不一定最优。

先看可行性条件，由于B和$\mathbf{b}$都没有变化，$B^{-1}\mathbf{b} \geq 0$仍然成立，所以可行性不受影响，原来的最优解仍然可行；

再看最优性条件，c_j或$\mathbf{c}_{BV}$的变化都会影响$\bar{c}_j = \mathbf{c}_{BV}B^{-1}\mathbf{a}_j - c_j$的结果，所以最优性受影响。如果改变后的$\bar{c}_j \geq 0, \forall j \in NBV$仍然成立，则原最优解仍然最优；反之原最优解可行但不再最优。

2. 约束右端常量变化

$\mathbf{b}$变化，此时原最优解要么仍然是最优解，要么不再可行。

$\mathbf{b}$的变化使得可行性条件受影响，最优性条件不受影响。因此，如果改变后可行性条件$B^{-1}\mathbf{b} \geq 0$仍然成立，那么原最优解仍然最优；如果不成立，那么原最优解不再可行，需要重新寻找最优解。

3. 约束系数变化

（1）非基本变量的约束系数N变化

此时原最优解一定可行，但不一定最优。可行性条件不受影响，最优性条件受影响。与之间的分析相同，如果改变后的$\bar{c}_j \geq 0, \forall j \in NBV$仍然成立，则原最优解仍然最优（但注意此时目标函数值会改变）；反之原最优解可行但不再最优。

（2） 基本变量的约束系数B变化

此时原最优解不可行/可行非最优/最优三种情况都可能。因为两个条件都受到了影响，需要重新计算两个条件是否满足。

本节介绍了「敏感度分析」，借助单纯形法和对偶理论，当线性规划中的参数发生变化时，不一定需要重新求解问题，可以通过计算可行性条件和最优性条件的成立情况来判断原最优解是否仍然可行/最优。

到这里第四章内容全部结束。简单回顾，在这一章中，我们先引入了「对偶问题」，介绍它的优势和写法；再次，通过「对偶原理」，即弱对偶定理、强对偶定理和互补松弛定理，进一步阐述了原问题和对偶问题的解之间的关系；基于对偶原理，发现利用对偶可行性也可以逐步搜索出原始问题最优解，产生了另一种线性规划求解算法——「对偶单纯形法」；最后，当线性规划的参数发生变动时，通过「敏感度分析」可以判断参数变化对最优解的影响情况。

本章参考文献

[1] L Winston, Wayne & B Goldberg, Jeffrey. (2004). Operations research: applications and algorithms.

[2] Frederick S. Hillier, Gerald J. Lieberman. Introduction to Operations Research (9th).

[3] Vijay V. Vazirani. (2003). Approximation Algorithms.

[4] 为什么我们要考虑线性规划的对偶问题？-知乎https://www.zhihu.com/question/26658861

第5章 大规模线性规划求解算法

经典的大规模线性规划求解算法有切平面法，列生成和Dantzig-Wolfe分解等，很多算法的应用场景都属于整数规划的范畴。在线性规划专题下，我们简单地给读者介绍列生成和Dantzig-Wolfe分解的原理，想要深入了解的读者可以阅读整数规划专题的相关内容。

5.1 列生成算法

作者：阎泳楠，同济大学 交通运输工程 硕士研究生

研究方向：交通网络优化与建模

列生成算法是基于单纯形法的基本思想改进而来的，要理好解列生成，前提条件是掌握好线性规划单纯形法和对偶问题等知识，主要是单纯形法中非基变量进基时检验数（reduced cost）的计算，对偶问题中的影子价格和对偶变量。遗忘了或者对这部分内容不熟悉的读者，可以往前阅读相关章节的内容。

5.1.1 引例

下面通过木料切割（Cutting Stock Problem）这一经典问题，介绍列生成算法。

引例：木材厂有3种不同长度的木料：9m,14m,和16m。这三种木料的成本价分别是5，9和10元。有顾客需要4m的木料30根，5m的木料20根以及7m的木料40根。如何切割木料，使得既能满足顾客需求，又能使成本最少？

对这样一个小型算例，我们不妨试着枚举一下，这三种长度的木料（9,14,16）一共可以有多少种满足顾客需求（4,5,7）的切割方式。图5.1列出了13种可行的切割方案A_j，A_j是个三维的列向量，表示每个方案关于三种需求样式的切割次数，每个切割方案的数量用x_j表示（注意，此处仅考虑物料浪费最少的切割方案，故并非所有方案）。L表示每种切割方案所用原材料的长度：L=(9,14,16)T，而C对应每种切割方案的成本：C=(5,9,10)T，与原材料长度相关；l表示不同的需求样式：l=(4,5,7)T；d表示不同样式的需求量：d=(30,20,40)T。我们举个例子，切割方案 = (2, 0, 0)T表示原材料9m长的木棒，可被切割成2个4m长的木料。

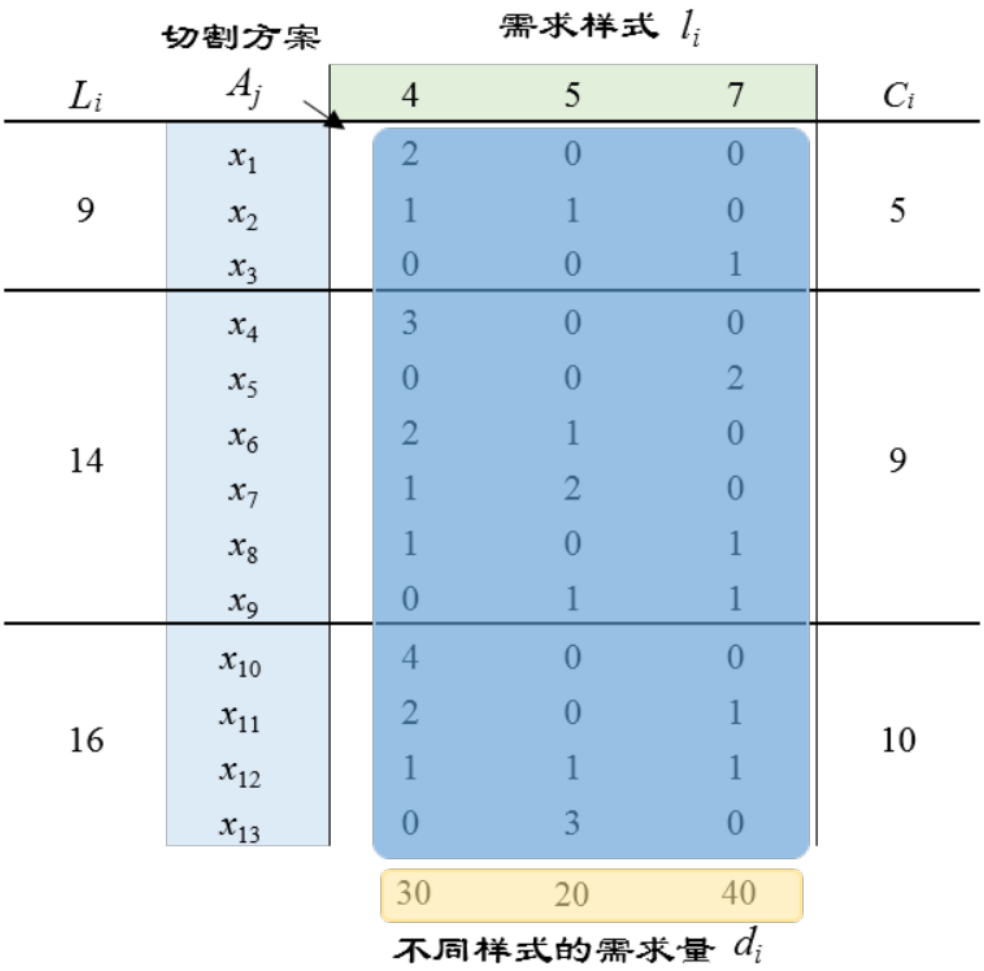


图5.1 满足需求的切割方案

对该物料切割问题建立线性规划模型：

式（5.1）

$$\min \sum_{i=j}^{13} c_j x_j$$

$$\text{s.t.} \quad \sum_{i=j}^{13} a_{ij} x_j \geq d_i \quad i \in \{1,2,3\}$$

$$x_j \in Z^+ \quad j \in \{1,2,\ldots,13\}$$

式中，x_j表示第j种方案的数量，取值为正整数；a_{ij}表示第j种方案中需求样式i的个数，其中i取值为1，2，3，分别表示4m、5m和7m的需求样式。

我们依然能用单纯形法求解上述模型。先不谈单纯形法求解该模型的效率，仅从需要枚举所有的切割方案x_j这一操作，就能知道上述求解方法一定不是最优的。而且上述枚举法存在以下缺点：

- 当原材料长度增加时，切割方案呈指数式增长，我们无法枚举所有的x_j；
- 枚举出来的切割方案可能会剩余过多的木料，这种方案通常不会被选择。

单纯形法要求我们列出所有的变量（x_j）。回忆一下，单纯形法在每次迭代的过程中，只能用一个非基变量替换一个基变量，而最终的基变量个数仅与约束条件的数量有关。这么看来，有大部分的x_j是不会被用到的（也就是说这些方案都不够好）。因此，我们不需要，也没必要列出所有的x_j；再者，对于大规模的问题，我们无法列出满足条件的所有变量。基于此，有学者就提出了求解大规模线性规划的高效算法 —— 列生成法，主要适用于变量数目（列）大于约束条件个数（行）的线性规划问题。

5.1.2 列生成

列生成算法的主要思想是对那些无法列出所有变量的原问题（master problem，MP），先考虑有限变量的主问题restricted master problem (RMP)，其余变量在有需要的时候，再添加到RMP中。什么变量是需要被加入到RMP中的呢？在单纯形法中，我们根据非基变量检验数的正负来选择进基变量；在列生成中，我们通过子问题（subproblem）来寻找满足reduced cost条件的变量，如果找到的话，就将它加入到RMP中，直至找不到为止，那么MP就求得了最优解。

对5.1.1的引例，我们已经列出了包含13种切割方案的master problem，即式(5.1)。假设初始只考虑3种切割方案(可以与图5.1中列举的方案不一致，如仅考虑9m长原材料的以下三种切割样式x_1、x_2和 x_3,分别取A_1=(2,0,0)T、A_2=(0,1,0)T和A_3=(0,0,1)T，可得到RMP(5.2)。在式(5.2)中，变量的整数约束松弛成了非负约束。

RMP(5.2):

$$
\begin{aligned}
&\min 5x_1 + 5x_2 + 5x_3 \\
&\text{s.t.} \quad 2x_1 \geq 30 \\
&x_2 \geq 20 \\
&x_3 \geq 40 \\
&x_1, x_2, x_3 \geq 0
\end{aligned}
$$

Subproblem的构建依赖于求解RMP得到的对偶变量，也就是影子价格**w**。获得对偶变量有两种方法：一是通过求RMP的对偶问题获得，二是通过RMP问题中的$C_B B^{-1}$计算得到。基于检验数的计算公式，可以得到Subproblem的目标函数，即

$$
\max \sigma_j = z_j - c_j = C_B B^{-1} A_j - c_j = \sum_{i=1}^{m} w_i a_{ij} - c_j \quad i \in \{1, 2, 3\}
$$

若$max\ \sigma_j > 0$，则A_j可以加到RMP的约束条件中，变量加x_j到目标函数中；若 $max\ \sigma_j \leq 0$，则RMP的当前解已是最优解。

Subproblem的约束条件如下:

$$\text{s.t.} \quad \sum_{i=1}^{3} l_i a_{ij} \le L_i$$

$$a_i \ge 0 \quad x_j \in Z^+$$

不难理解，影子价格**w**指的是满足顾客需求可获得的价值，那么子问题目标函数的含义可以理解为新切割方案带来的利润，所以只要使利润大于0的切割方案才能被加到RMP中。子问题的个数一般取决于原材料的类型，求解一个子问题能够得到一个切割方案A_j。约束条件表示当前切割方案要足原材料的要求。Cutting Stock Problem这类问题的subproblem其实是个背包问题，可以用动态规划等方法求解。对车辆路径问题（Vehicle Routing Problem, VRP）而言，它的子问题是最短路问题。可见，列生成算法可以将规模庞大的原问题分解成规模较小的RMP以及一个经典的，有成熟求解算法的线性规划子问题。求解这两个问题的难度将远远小于求解原问题的难度。

接下来，我们用列生成算法求解引例。

Iteration 1：基于RMP(5-2), 用单纯形法可求得X^* = (15,20,40)T, Z^* = 375，影子价格 **w** = $C_B B^{-1}$ = (2.5,5,5). 设新加入的列为A_4 = (a_{14},a_{24},a_{34})T，构造Subproblem。本算例中有三种不同长度的原材料L=(9, 14, 16)T，对应三种不同的成本C=(5,9,10)T，因此这里有三个子问题。

Subproblem 1:

$$\max 2.5a_{14} + 5a_{24} + 5a_{34} - \mathbf{5}$$
$$\text{s.t.} \quad 4a_{14} + 5a_{24} + 7a_{34} \le \mathbf{9}$$
$$a_{ij} \in Z^+$$

Subproblem 2:

$$\max 2.5a_{14} + 5a_{24} + 5a_{34} - \mathbf{9}$$
$$\text{s.t.} \quad 4a_{14} + 5a_{24} + 7a_{34} \le \mathbf{14}$$
$$a_{ij} \in Z^+$$

Subproblem 3:

$$\max 2.5a_{14} + 5a_{24} + 5a_{34} - \mathbf{10}$$
$$\text{s.t.} \quad 4a_{14} + 5a_{24} + 7a_{34} \le \mathbf{16}$$
$$a_{ij} \in Z^+$$

求得三个子问题的解如下：

Subproblem 1: A_4 = (1,1,0)T , $\sigma_4 = 2.5.$

Subproblem 2: A_4 = (1,2,0)T , $\sigma_4 = 3.5.$

Subproblem 3: A_4 = (0,3,0)T , $\sigma_4 = 5$.

三个子问题的检验数都大于0，我们取检验数最大的子问题的解 A_4 = (0,3,0)T加入RMP(5.2)中。

Iteration 2： 式(5-2)加入了新的切割方案A_4后，得到新的RMP(5.3)如下：

RMP(5.3):

$$\begin{gathered}\min 5x_1 + 5x_2 + 5x_3 + \mathbf{10}x_4 \\ \text{s.t.} \quad 2x_1 \geq 30 \\ x_2 + \mathbf{3}x_4 \geq 20 \\ x_3 \geq 40 \\ x_1, x_2, x_3, x_4 \geq 0\end{gathered}$$

切割方案A_4对应原材料为16m的木棒，切割成本为10元。可求得X^* = (15,0,40,20/3)T, Z^* = 341.7，影子价格 **w** =$C_B B^{-1}$ = (2.5,10/3,5). 设新加入的列为A_5 = (a_{15},a_{25},a_{35})T ,构造Subproblem.

Subproblem:

$$\begin{gathered}\max \quad 2.5a_{15} + 10/3a_{25} + 5a_{35} - c_j \\ \text{s.t.} \quad 4a_{15} + 5a_{25} + 7a_{35} \leq L_i \\ a_{ij} \in Z^+\end{gathered}$$

第二次迭代中同样有三个子问题，其中$c_j \in \{5, 9, 10\}$,$L_i \in \{9, 14, 16\}$.

求得三个子问题的解如下：

Subproblem 1: A_5 = (0,0,1)T , $\sigma_5 = 0$.

Subproblem 2: A_5 = (0,0,2)T , $\sigma_5 = 1$.

Subproblem 3: A_5 = (0,0,3)T , $\sigma_5 = 0$.

三个检验数中只有Subproblem 2的检验数大于0，故将 A_5 = (0,0,2)T加入到新的RMP中。

Iteration 3： 式(5.3)加入了新的切割方案A_5后，得到RMP(5.4)如下：

RMP(5.4):

$$\min 5x_1 + 5x_2 + 5x_3 + 10x_4 + \mathbf{9}x_5$$
$$\text{s.t.} \quad 2x_1 \geq 30$$
$$x_2 + 3x_4 \geq 20$$
$$x_3 + \mathbf{2}x_5 \geq 40$$
$$x_1, x_2, x_3, x_4, x_5 \geq 0$$

切割方案A_5对应原材料为14m的木棒，切割成本为9元。可求得X^* = (15,0,0,20/3,20)T, Z^* = 322，影子价格 **w** =$C_B B^{-1}$ = (2.5,10/3,4.5). 设新加入的列为A_6 = (a_{16},a_{26},a_{36})T，构造Subproblem.

Subproblem:

$$\max \quad 2.5a_{16} + 10/3a_{26} + 4.5a_{36} - c_j$$
$$\text{s.t.} \quad 4a_{16} + 5a_{26} + 7a_{36} \leq L_i$$
$$a_{ij} \in Z^+$$

求得三个子问题的解如下：

Subproblem 1: A_6 = (1,1,0)T , $\sigma_6 = 5/6.$

Subproblem 2: A_6 = (1,2,0)T , $\sigma_6 = 1/6.$

Subproblem 3: A_6 = (0,3,0)T , $\sigma_6 = 0.$

三个检验数中Subproblem 1的检验数最大，将该检验数对应的切割方案 A_6 = (1,1,0)T加入到新的RMP中。

Iteration 4： 式(5.4)加入了新的切割方案A_6后，得到新的RMP(5.5)如下：

RMP(5.5):

$$\min 5x_1 + 5x_2 + 5x_3 + 10x_4 + 9x_5 + \mathbf{5}x_6$$
$$\text{s.t.} \quad 2x_1 + \mathbf{1}x_6 \geq 30$$
$$x_2 + 3x_4 + \mathbf{1}x_6 \geq 20$$
$$x_3 + 2x_5 \geq 40$$
$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$$

切割方案A_6对应原材料为9m的木棒，切割成本为5元。可求得X^* = (5,0,0,0,20,20)T, Z^* = 305，影子价格 **w** =$C_B B^{-1}$ = (2.5, 2.5 ,4.5). 设新加入的列为A_7 = (a_{17},a_{27},a_{37})T，构造Subproblem.

Subproblem:

$$\begin{aligned} \max \quad & 2.5a_{17} + 2.5a_{27} + 4.5a_{37} - c_j \\ \text{s.t.} \quad & 4a_{17} + 5a_{27} + 7a_{37} \le L_i \\ & a_{ij} \in Z^+ \end{aligned}$$

求得三个子问题的解如下：

Subproblem 1: A_7 = (0,0,1)T , $\sigma_7 = -0.5$.

Subproblem 2: A_7 = (0,0,2)T , $\sigma_7 = 0$.

Subproblem 3: A_7 = (0,0,2)T , $\sigma_7 = -1$.

当前子问题的检验数均小于等于0，所以X* = (5,0,0,0,20,20)T是RMP(5.5)的最优解。具体方案如下：方案①：将5根9m长的原材料各切成2个4m,；方案②：将20根14m长的原材料各切成2个7m；方案③：将20根9m长的原材料各切成一个4m和一个5m。

原材料长度 L	方案选择个数 x_j	需求样式 1		
		4	5	7
9	5	2	0	0
14	20	0	0	2
9	20	1	1	0
sum=样式总需求d		30	20	40

切割方案 a_{ij}

图5.2 切方案

5.1.3 延伸

除了经典的Cutting Stock Problem，旅行商问题（Travelling salesman problem, TSP），车辆路径问题（Vehicle Routing Problem, VRP），车间调度问题（scheduling）等都可以用列生成算法求得下界。本算例的计算过程都是手动计算的，如果有读者想要了解关于调用求解器求解列生成的算例，可查询参考文献[1-3]。

5.2 Dantzig-Wolfe分解

作者：李崇楠 北京交通大学 交通运输规划与管理 研究生在读

研究方向：运输组织优化

在本部分，我们考虑求解具有特殊结构的线性规划问题。特别地，我们考虑具有方块对角（Block Angular）的线性规划问题，并使用Dantzig-Wolfe分解方法来解决这样的问题。

5.2.1 方块对角线性规划问题的分解

考虑具有如下形式的线性规划问题：

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

其中$\mathbf{A}$是一个$m \times n$维度的矩阵，并且假设该矩阵能够写成如下的形式：

$$\mathbf{A} = \begin{bmatrix} L_1 & L_2 & \cdots & L_K \\ A_1 & & & \\ & A_2 & & \vdots \\ & & \ddots & \\ & & & A_K \end{bmatrix}$$

其中L_k是一个子矩阵，维度为$m_L \times n_k$，$k = 1, \cdots, K$，并且A_k是一个子矩阵，维度为$m_k \times n_k$，$k = 1, \cdots, K$，并且满足：$\sum_{k=1}^{K} n_k = n$以及$m_L + \sum_{k=1}^{K} m_k = m$。这样的矩阵$\mathbf{A}$称为方块对角矩阵.。

令$\mathbf{x}^k$, $\mathbf{c}^k$和$\mathbf{b}^k$为与$\mathbf{A}$对应的子向量，满足：

$$\mathbf{x}=\begin{bmatrix}\mathbf{x}^1\\ \vdots\\ \mathbf{x}^K\end{bmatrix},\quad \mathbf{c}=\begin{bmatrix}\mathbf{c}^1\\ \vdots\\ \mathbf{c}^K\end{bmatrix},\quad \mathbf{b}=\begin{bmatrix}\mathbf{b}^1\\ \vdots\\ \mathbf{b}^K\end{bmatrix}.$$

进而，原来的线性规划问题可以写成如下的形式：

$$\begin{aligned}\min\quad & (\mathbf{c}^1)^T\mathbf{x}^1+(\mathbf{c}^2)^T\mathbf{x}^2+\cdots+(\mathbf{c}^K)^T\mathbf{x}^K\\ \text{s.t.}\quad & L_1\mathbf{x}^1+L_2\mathbf{x}^2+\cdots+L_K\mathbf{x}^K\le\mathbf{b}^0\\ & A_1\mathbf{x}^1\le\mathbf{b}^1\\ & A_2\mathbf{x}^2\le\mathbf{b}^2\\ & \vdots\\ & A_K\mathbf{x}^K\le\mathbf{b}^K\\ & \mathbf{x}^1,\mathbf{x}^2,\cdots,\mathbf{x}^K\ge\mathbf{0}\end{aligned}$$

或者使用求和算符写成更加紧凑的形式：

$$\begin{aligned}\min\quad & \sum_{k=1}^{K}(\mathbf{c}^k)^T\mathbf{x}^k\\ \text{s.t.}\quad & \sum_{k=1}^{K}L_k\mathbf{x}^k\le\mathbf{b}^0\\ & A_k\mathbf{x}^k\le\mathbf{b}^k,\quad \forall k=1,\cdots,K.\\ & \mathbf{x}^k\ge\mathbf{0},\quad \forall k=1,\cdots,K.\end{aligned}$$

约束$\sum_{k=1}^{K}L_k\mathbf{x}^k\le\mathbf{b}^0$称为耦合约束（coupling constraints）或链接约束（linking constraints），这种约束的个数为$\mathbf{b}^0$向量的行数，即之前提到的m_L。之所以称为耦合约束，是因为如果没有这些约束，那么原来的线性规划问题就可以等价地分解为K个独立的子问题（subproblem），其中第k个子问题是：

$$(SP_k):\begin{cases}\min & (\mathbf{c}^k)^T\mathbf{x}^k \\ \text{s.t.} & A_k\mathbf{x}^k \le \mathbf{b}^k \\ & \mathbf{x}^k \ge \mathbf{0}\end{cases}$$

这个子问题也是一个线性规划（linear programming）。

分解的基本思想是寻找像刚才提到的这种具有方块对角形式的线性规划问题，这时整个问题并不必须一次性全部解决，而是可以分解为若干更小更好解决的子问题。原问题经过**Dantzig-Wolfe**分解后，会生成一个只带有耦合约束的主问题（**master problem**）与若干子问题。

主问题可以写成如下形式：（注意没有非负约束！）

$$(MP):\begin{cases}\min & \sum_{k=1}^{K}(\mathbf{c}^k)^T\mathbf{x}^k \\ \text{s.t.} & \sum_{k=1}^{K}L_k\mathbf{x}^k \le \mathbf{b}^0\end{cases}$$

5.2.2 主问题再塑造

仅仅将问题分解为如上介绍的形式是不够的。为了使得分解后的求解更加高效，主问题需要进行再塑造（reformulation）。再塑造可以保证主问题与子问题在求解的过程中交换信息，同时保证分别对每个问题进行求解。但是在重塑后的主问题中，约束矩阵的列数会远大于行数，增加求解的难度。以下内容是主问题再塑造的详细过程，对推导不感兴趣的读者可以略过本节，不影响后续阅读。

再塑造的关键是如下事实：子问题都是线性规划问题，因此每个子问题的可行域都是多边形（polyhedron）。而在单纯形算法学习中，我们学习过多边形的表示定理（Resolution Theorem）：可行域内任意一点可以表示为极点的凸组合和极方向的非负组合，用符号表示该定理为：

设$\mathbf{x}^1,\mathbf{x}^2,\cdots,\mathbf{x}^N$为某一线性规划可行域的极点（extreme point），$\mathbf{d}^1,\mathbf{d}^2,\cdots,\mathbf{d}^M$为可行域的极方向（extreme direction）,那么可行域内的任意一点$\mathbf{x}$可以用下式表示：

$$\mathbf{x}=\sum_{i=1}^{N}\lambda_i\mathbf{x}^i+\sum_{j=1}^{M}\mu_j\mathbf{d}^j$$

其中$\lambda_i \in [0,1], \forall i=1,\cdots,N.\ \sum_{i=1}^{N}\lambda_i=1$且$\mu_j \geq 0, \forall j=1,\cdots,M.$令$P_k=\{\mathbf{x}^k | A_k\mathbf{x}^k \leq \mathbf{b}^k, \mathbf{x}^k \geq \mathbf{0}\}$为第$k$个子问题$(SP_k)$的可行域. 令$\mathbf{v}_1^k, \mathbf{v}_2^k, \cdots, \mathbf{v}_{N_k}^k$为$P_k$的极点，$\mathbf{d}_1^k, \mathbf{d}_2^k, \cdots, \mathbf{d}_{l_k}^k$为$P_k$的极方向。根据表示定理，$P_k$中的任意一点$\mathbf{x}^k$可以表示为

$$\mathbf{x}^k=\sum_{i=1}^{N_k}\lambda_i^k\mathbf{v}_i^k+\sum_{j=1}^{l_k}\mu_j^k\mathbf{d}_j^k$$

其中，$\sum_{i=1}^{N_k}\lambda_i^k=1$, 且对于$\forall i=1,\cdots,N_k,\ \lambda_i^k \geq 0$, 对于$\forall j=1,\cdots,l_k, \mu_j^k \geq 0$。将$\mathbf{x}^k$由表示定理得到的展开式代入主问题便得到：

$$(MP): \begin{cases} \min & \sum_{k=1}^{K}(\mathbf{c}^k)^T(\sum_{i=1}^{N_k}\lambda_i^k\mathbf{v}_i^k+\sum_{j=1}^{l_k}\mu_j^k\mathbf{d}_j^k) \\ \text{s.t.} & \sum_{k=1}^{K}L_k(\sum_{i=1}^{N_k}\lambda_i^k\mathbf{v}_i^k+\sum_{j=1}^{l_k}\mu_j^k\mathbf{d}_j^k) \leq \mathbf{b}^0 \\ & \sum_{i=1}^{N_k}\lambda_i^k=1, \quad k=1,\cdots,K. \\ & \lambda_i^k \geq 0, \quad i=1,\cdots,N_k, k=1,\cdots,K. \\ & \mu_j^k \geq 0, \quad j=1,\cdots,l_k, k=1,\cdots,K. \end{cases}$$

将括号打开，得到：

$$(MP): \begin{cases} \min & \sum_{k=1}^{K}\sum_{i=1}^{N_k}\lambda_i^k(\mathbf{c}^k)^T(\mathbf{v}_i^k)+\sum_{k=1}^{K}\sum_{j=1}^{l_k}\mu_j^k(\mathbf{c}^k)^T(\mathbf{d}_j^k) \\ \text{s.t.} & \sum_{k=1}^{K}\sum_{i=1}^{N_k}\lambda_i^k(L_k\mathbf{v}_i^k)+\sum_{k=1}^{K}\sum_{j=1}^{l_k}\mu_j^k(L_k\mathbf{d}_j^k) \leq \mathbf{b}^0 \\ & \sum_{i=1}^{N_k}\lambda_i^k=1, \quad k=1,\cdots,K. \\ & \lambda_i^k \geq 0, \quad i=1,\cdots,N_k, k=1,\cdots,K. \\ & \mu_j^k \geq 0, \quad j=1,\cdots,l_k, k=1,\cdots,K. \end{cases}$$

这里我们认为极点$\mathbf{v}_i^k$和极方向$\mathbf{d}_j^k$是已知信息，需要求解出最优的系数λ_i^k与μ_j^k。再由表示定理推算出最优解$\mathbf{x}^*$。换句话说，此时原问题的决策变量是λ_i^k与μ_j^k。

对于每一个极点$\mathbf{v}_i^k \in P_k$，令$f_i^k = (\mathbf{c}^k)^T(\mathbf{v}_i^k)$和$q_i^k = L_k\mathbf{v}_i^k$，对于每一个极方向$\mathbf{d}_j^k$，令$f_j^{-k} = (\mathbf{c}^k)^T(\mathbf{d}_j^k)$和$q_j^{-k} = L_k\mathbf{d}_j^k$。

代入主问题得到：

$$(MP): \begin{cases} \min & \displaystyle\sum_{k=1}^{K}\sum_{i=1}^{N_k}\lambda_i^k f_i^k + \sum_{k=1}^{K}\sum_{j=1}^{l_k}\mu_j^k f_j^{-k} \\ \text{s.t.} & \displaystyle\sum_{k=1}^{K}\sum_{i=1}^{N_k}\lambda_i^k q_i^k + \sum_{k=1}^{K}\sum_{j=1}^{l_k}\mu_j^k q_j^{-k} \le \mathbf{b}^0 \\ & \displaystyle\sum_{i=1}^{N_k}\lambda_i^k = 1, \quad k = 1,\cdots,K. \\ & \lambda_i^k \ge 0, \quad i = 1,\cdots,N_k, k = 1,\cdots,K. \\ & \mu_j^k \ge 0, \quad j = 1,\cdots,l_k, k = 1,\cdots,K. \end{cases}$$

其中约束$\sum_{i=1}^{N_k}\lambda_i^k = 1, \quad k = 1,\cdots,K$称为对应于子问题$(SP_k)$的凸约束（convexity constraint）。

主问题可以进一步紧凑表示：

$$(MP): \begin{cases} \min & \mathbf{f}_v^T\boldsymbol{\lambda} + \mathbf{f}_d^T\boldsymbol{\mu} \\ \text{s.t.} & \mathbf{Q}_v\boldsymbol{\lambda} + \mathbf{Q}_d\boldsymbol{\mu} + \mathbf{s} = \mathbf{r} \\ & \boldsymbol{\lambda} \ge \mathbf{0}, \boldsymbol{\mu} \ge \mathbf{0}, \mathbf{s} \ge \mathbf{0} \end{cases}$$

其中

$$\boldsymbol{\lambda} = (\lambda_1^1,\cdots,\lambda_{N_1}^1,\lambda_1^2,\cdots,\lambda_{N_2}^2,\cdots,\lambda_1^K,\cdots,\lambda_{N_K}^K)^T$$

$$\boldsymbol{\mu} = (\mu_1^1,\cdots,\mu_{l_1}^1,\mu_1^2,\cdots,\mu_{l_2}^2,\cdots,\mu_1^K,\cdots,\mu_{l_K}^K)^T$$

$$\boldsymbol{f}_v = (f_1^1,\cdots,f_{N_1}^1,f_1^2,\cdots,f_{N_2}^2,\cdots,f_1^K,\cdots,f_{N_K}^K)^T$$

$$\boldsymbol{f}_d = (f_1^{-1},\cdots,f_{l_1}^{-1},f_1^{-2},\cdots,f_{l_2}^{-2},\cdots,f_1^{-K},\cdots,f_{l_K}^{-K})^T$$

$$\mathbf{r}^T = [(\mathbf{b}_0)^T, \mathbf{e}^T] = [(\mathbf{b}_0)^T, \underbrace{(1,\cdots,1)^T}_{K}]$$

$$\mathbf{s}^T = [\underbrace{(\mathbf{s}^0)^T}_{m_L}, \underbrace{(0,\cdots,0)^T}_{K}]$$

$\mathbf{Q}_v$是一个矩阵，它对应于λ_i^k的列为：

$$\begin{bmatrix}\mathbf{q}_i^k \\ \mathbf{e}_k\end{bmatrix} = \begin{bmatrix}L_k\mathbf{v}_i^k \\ \mathbf{e}_k\end{bmatrix}$$

$\mathbf{e}_k$为单位向量，在第k处分量为 1。$\mathbf{Q}_d$是一个矩阵，它对应于μ_j^k的列为：

$$\begin{bmatrix}\mathbf{q}_j^{-k} \\ \mathbf{0}\end{bmatrix} = \begin{bmatrix}L_k\mathbf{d}_j^k \\ \mathbf{0}\end{bmatrix}$$

由于子问题的可行域可以有大量的极点和极方向，即使是中等规模的问题，主问题决策变量λ_i^k与μ_j^k的数目可能也是海量的。换句话说，在重塑后的主问题中，约束矩阵的列数会远大于行数。

5.2.3 限制主问题和修正单纯形法

针对上述重塑主问题带来的决策变量数量及其多这一难点，下面将介绍修正单纯形法（revised simplex method）求解主问题。修正单纯形法的好处是求解过程中大量的决策变量会置零（亦即非基变量），这就使得没有必要去生成整个重塑主问题。

基于上述分析，启发我们去建立一个更小版本的主问题，称为限制主问题（restricted master problem），只有一小部分对应于当前基本可行解的λ_i^k与μ_j^k会在此问题中，其他决策变量是非基的，取值为零。如果在当前基本可行解下，非基变量的检验数（reduced cost）都是非负的，那么修正单纯形法终止，得到最优解，否则会有某个检验数为负数的非基变量进基。

假设当前限制主问题的基为$\mathbf{B}$，令$\boldsymbol{\pi}^T = \mathbf{f}_B^T\mathbf{B}^{-1}$，其中$\mathbf{f}_B$由与基变量$\lambda_i^k$、$\mu_j^k$对应的$f_i^k$、$f_j^{-k}$组成.假定$\boldsymbol{\pi}$的元素为如下形式：

$$\boldsymbol{\pi} = \begin{bmatrix}\boldsymbol{\pi}^1 \\ \pi_1^2 \\ \pi_2^2 \\ \vdots \\ \pi_K^2\end{bmatrix}$$

其中$\boldsymbol{\pi}^1$是对应耦合约束的对偶变量，π_i^2是在限制主问题中对应于子问题SP_i的凸约束的对偶变量，进而，非基变量λ_i^k的检验数为

$$r_i^k = f_i^k - \boldsymbol{\pi}^T \begin{bmatrix} \boldsymbol{q}_i^k \\ \\ \mathbf{e}_k \end{bmatrix} = (\mathbf{c}^k)^T(\mathbf{v}_i^k) - (\boldsymbol{\pi}^1)^T L_k \boldsymbol{v}_i^k - \pi_2^k$$

非基变量μ_j^k的检验数为

$$r_j^{-k} = f_j^{-k} - \boldsymbol{\pi}^T \begin{bmatrix} \boldsymbol{q}_j^{-k} \\ \\ \mathbf{0} \end{bmatrix} = (\mathbf{c}^k)^T(\mathbf{d}_j^k) - (\boldsymbol{\pi}^1)^T L_k \boldsymbol{d}_j^k$$

这里会有大量的非基变量，但是没有必要计算所有的检验数.事实上，计算检验数中最小的就足够了。令

$$r_{\min} = \min_{k=1,\cdots,K} \left\{ \min_{i=1,\cdots,N_k} \{r_i^k\} \right\}$$

或

$$r_{\min} = \min_{k=1,\cdots,K} \left\{ \min_{i=1,\cdots,N_k} \{(\mathbf{c}^k)^T(\mathbf{v}_i^k) - (\boldsymbol{\pi}^1)^T L_k \mathbf{v}_i^k - \pi_2^k\} \right\}$$

我们进一步令 $r_*^k = \min_{i=1,\cdots,N_k}\{r_i^k\}$，然后让$r_*^k$作为子问题$SP_k$的目标函数：

$$(SP_k): \begin{cases} \min & \sigma_k = \left((\mathbf{c}^k)^T - (\boldsymbol{\pi}^1)^T L_k\right)\mathbf{x}^k \\ \\ \text{s.t.} & A_k \mathbf{x}^k \le \mathbf{b}^k \\ \\ & \mathbf{x}^k \ge \mathbf{0} \end{cases}$$

注意到目标函数没有项$-\pi_k^2$，这是因为在固定k的情况下，$-\pi_k^2$这一项是固定的。

假定使用修正单纯形法去求解子问题，所以如果子问题是有界的（bounded），那么生成一个最优解$\mathbf{x}^k$，且它为可行域其中一个极点$\mathbf{v}_i^k$。我们使用符号$i^* \in \{1,\cdots,N_k\}$来代表最优极点的标号，那么最优极点为$\mathbf{v}_{i^*}^k$。另外，使用σ_k^*表示子问题SP_k的最优目标函数值，进而有$r_*^k = \sigma_k^* - \pi_2^k$。

下面讨论一下子问题在求解过程中可能出现的三种情况：

(1)如果所有子问题都是有界的，并且 $r_{\min} < 0$，令t是r_*^k取得最小的标号，即$r_{\min} = r_*^t$。对应于子问题SP_t的最优极点$\mathbf{v}_{i^*}^t$的列

$$\begin{bmatrix} \boldsymbol{q}_i^t \\ \\ \mathbf{e}_t \end{bmatrix} = \begin{bmatrix} L_t \boldsymbol{v}_{i^*}^t \\ \\ \mathbf{e}_t \end{bmatrix}$$

将会进入基$\mathbf{B}$(basis)。

(2)如果所有子问题都是有界的，并且$r_{\min} \geq 0$，那么当前的基为最优的基。

(3)如果这里有至少一个子问题是无界的（unbounded），令s是子问题无界的标号，即子问题SP_s无界.此时修正单纯形法对应于子问题SP_s会返回一个满足$\left((\mathbf{c}^s)^T - (\boldsymbol{\pi}^1)^T L_s\right)\mathbf{d}_{j^*}^s < 0$的极方向$\mathbf{d}_{j^*}^s$，其中 $j^* \in \{1, \cdots, l_s\}$，并且对应于$\mu_{j^*}^s$的列

$$\begin{bmatrix} \boldsymbol{q}_{j^*}^{-s} \\ \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} L_s \boldsymbol{d}_{j^*}^s \\ \\ \mathbf{0} \end{bmatrix}$$

将会进入基$\mathbf{B}$(basis)。

5.2.4 Dantzig-Wolfe分解方法的步骤

算法步骤

在经过前面的介绍之后，我们可以正式介绍Dantzig-Wolfe分解方法的步骤。

Step 0：初始化。生成主问题的一个初始基$\mathbf{B}$。令$\mathbf{x}_B$为基变量向量，$\bar{\boldsymbol{B}}$为基变量的指标集合（index set），并让所有非基变量为0，得到限制主问题。

Step 1：得到单纯形算子。通过解线性系统：$\mathbf{B}^T \boldsymbol{pi} = \mathbf{f}_B$来得到单纯形算子$\boldsymbol{\pi}$。

Step 2：最优性检验。对于每一个$k = 1, \cdots, K$，使用修正单纯形法求解子问题SP_k，亦即求解：

$$(SP_k): \begin{cases} \min \quad \sigma_k = \left((\mathbf{c}^k)^T - (\boldsymbol{\pi}^1)^T L_k\right)\mathbf{x}^k \\ \\ \text{s.t.} \quad A_k \mathbf{x}^k \leq \mathbf{b}^k \\ \\ \qquad \mathbf{x}^k \geq \mathbf{0} \end{cases}$$

如果 SP_k是无界的，那么进入Step 3；否则，令$\mathbf{x}^k = \mathbf{v}_{i^*}^k$来代表最优的基本可行解，并去计算检验数$r_*^k = \sigma_k^* - \pi_k^2$。如果所有的$k = 1, \cdots, K$都已经完成上面的工作后，考虑$r_{\min}$的情况：

若 $r_{\min} \geq 0$，那么终止算法，当前的基为最优的基，否则进入Step 3。

Step 3：列生成。如果所有子问题SP_k都是有界的，并且$r_{\min}<0$，那么令t为指标，该指标满足：$r_{\min}=r_*^t$。令

$$\bar{\boldsymbol{a}}=\begin{bmatrix}\boldsymbol{q}_{i^*}^t\\ \\ \mathbf{e}_t\end{bmatrix}=\begin{bmatrix}L_t\boldsymbol{v}_{i^*}^t\\ \\ \mathbf{e}_t\end{bmatrix}$$

其中$\mathbf{v}_{i^*}^t$是子问题SP_t的最优极点，并进入Step 4。

否则存在一个子问题SP_s是无界的，那么会有极方向$\mathbf{d}_{j^*}^s$生成，该极方向满足$((\mathbf{c}^s)^T-(\boldsymbol{\pi}^1)^TL_s)\mathbf{d}_{j^*}^s<0$，并且令

$$\bar{\boldsymbol{a}}=\begin{bmatrix}\boldsymbol{q}_{j^*}^{-s}\\ \\ \mathbf{0}\end{bmatrix}=\begin{bmatrix}L_s\boldsymbol{d}_{j^*}^s\\ \\ \mathbf{0}\end{bmatrix}$$

进入Step 4。

Step 4：生成下降方向。令下降方向为$\mathbf{d}$，求解线性系统$\mathbf{B}\mathbf{d}=-\bar{\boldsymbol{a}}$得到$\mathbf{d}$。如果$\mathbf{d}\geq\mathbf{0}$，那么原问题是无界的，终止算法，否则进入Step 5。

Step 5：生成步长。利用

$$\alpha=\min_{l\in\bar{\boldsymbol{B}}}\{-\frac{x_l}{d_l}|d_l<0\}$$

计算步长（最小比率检验 minimum ratio test），令l^*为指标满足

$$\alpha=-\frac{x_{l^*}}{d_{l^*}}$$

进入Step 6。

Step 6：更新基本解。$\mathbf{x}_B:=\mathbf{x}_B+\alpha\mathbf{d}$来更新基本解，进入Step 7。

Step 7：更新基。令$\mathbf{B}_{l^*}$为对应于出基变量x_{l^*}位于基$\mathbf{B}$的列。通过$\mathbf{B}_{l^*}$离开基，并在对应位置添加$\bar{\boldsymbol{a}}$的方式来更新基。回到Step 1。

参考代码下载

作者利用MATLAB完成了Dantzig – Wolfe分解的代码，代码下载地址为：

链接: https://pan.baidu.com/s/1NDnLFnMq4MjRUqAfNPIKiQ

密码: l2tz

参考文献

[1] 【学界/编码】从下料问题看整数规划中的列生成方法_附Gurobi求解器源代码

[2] 干货 | 10分钟带你彻底了解column generation（列生成）算法的原理附java代码

[3] 开源的Julia实例给出了column generation求解Cutting Stock Problem的详细步骤

第六章 网络流问题

作者：阎泳楠，同济大学 交通运输工程 硕士研究生

研究方向：交通网络优化与建模

网络流问题是一类特殊的线性规划问题，如前面章节讨论的运输问题，除了可以写成线性规划的形式，还能将该问题构建成网络的形式求解。我们的生活中有各种各样的网络，如道路网，电网，通信网等等。网络的应用也十分广泛，如物资的配送，线网的铺设，站点的选址，工程进度的安排等。本章将介绍基本的网络流概念以及四种重要的网络流问题：最短路径问题、最小生成树问题、最大流问题和最小费用流问题。

6.1 图论

网络流问题通常用图来表示。在本小节，我们主要介绍以下两个方面的内容：

（1）图论中的基本概念，定义本章节中出现的记号；

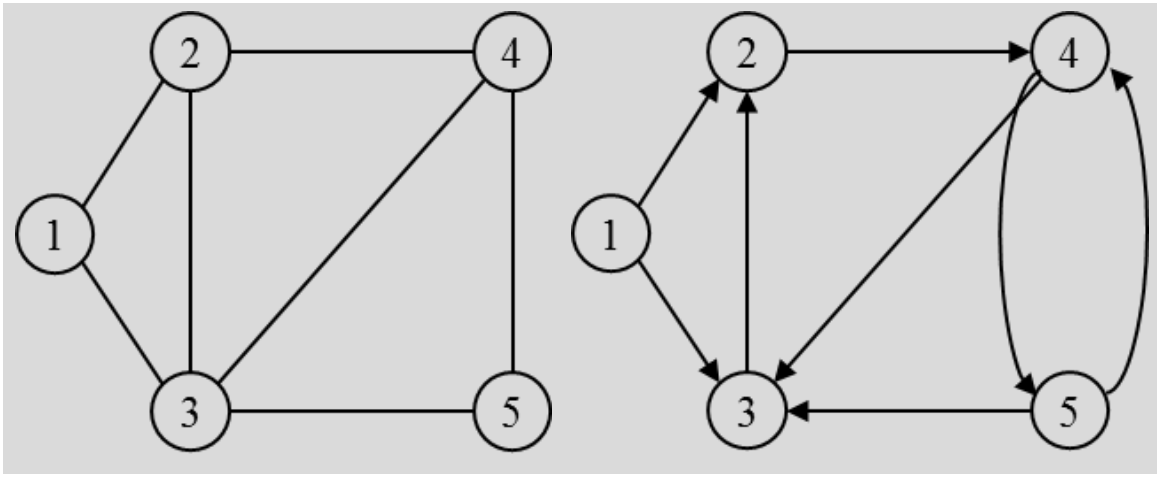
（2）介绍表征网络常用的数据结构。

前方预警，下文将涉及描述网络时不得不提到的术语，编者已尽力用最通俗易懂的语言去解释了。建议读者在阅读本节时可对照图示理解，有个印象即可，后续内容碰到了再来回顾相关概念。

6.1.1 图的基本概念

- 无向图和有向图

  网络，其实也就是一张图，由不同的节点和节点之间的连线（弧）组成。根据弧是否有方向，可分为有向弧和无向弧。若图中所有的弧都是无向弧（有向弧），则称为无向图$G=(N, E)$（有向图$G=(N, A)$）。N表示图中节点的集合，E表示无向弧，A表示有向弧。图6.1所示的两个网络都有五个节点，其中左图是无向图，有7条弧；右图是有向图，有8条弧。图论中的一些术语会因无向图和有向图而有所不同，下面将分别进行介绍。

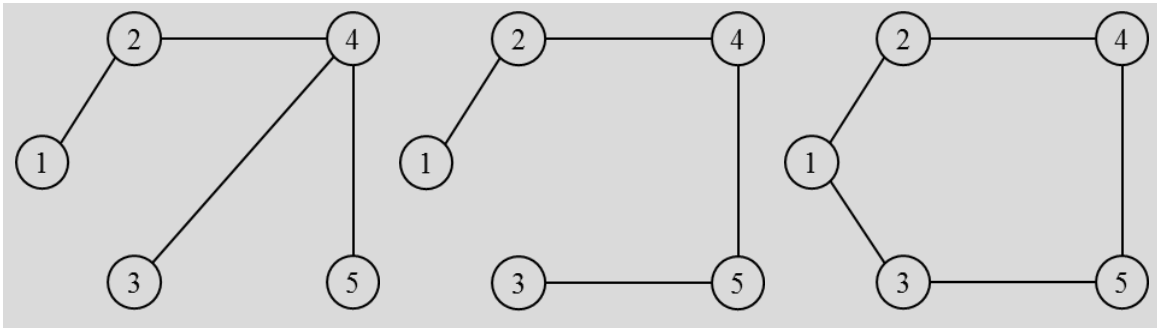


无向图　　有向图

图6.1 无向图和有向图

链和圈

在无向图中，如果两个节点不能直接相连，需要通过其它节点和弧连接，则称这两个节点之间的序列为一条链。如果一条链中没有重复的边，则为简单链；如果既无重复的边又无重复点，则称为初等链。始点与终点相同的链，称为圈。图6.2给出了图6.1中无向图中的一条简单链，初等链和圈。

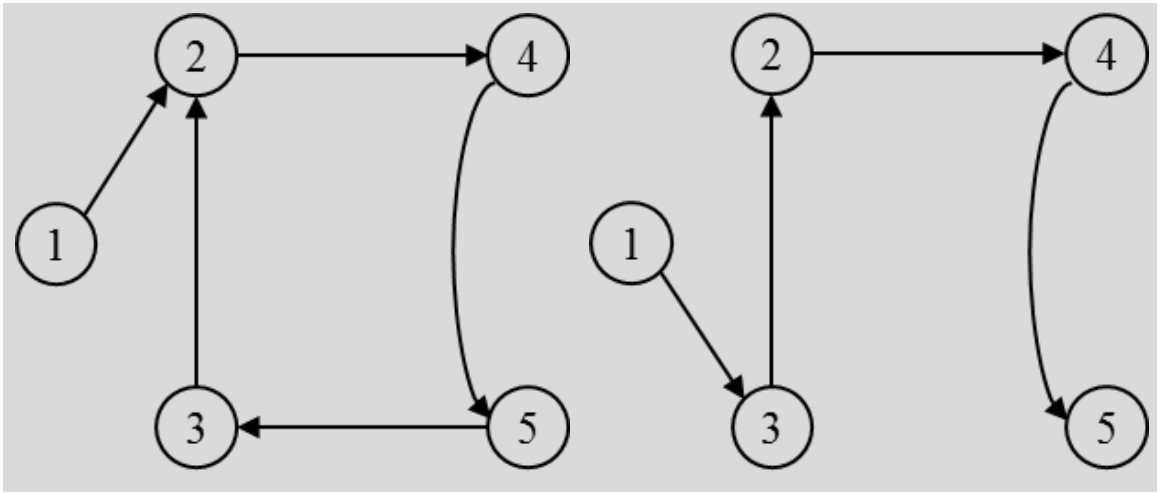


简单链　　初等链　　圈

图6.2 简单链、初等链和圈

- 路和回路

在有向图中，与链和圈相对应的概念是路和回路。路与链一样都是点序列，不同的是路具有方向性。同样地，一条路的起点与终点重合，则称为一条回路。如果路或回路上没有重复的点，则称它们为初等路和初等回路。图6.1中有向图的一条路，初等路如图6.3所示，该有向图不存在回路。



路：1-2-4-5-3-2　　初等路：1-3-2-4-5

图6.3 路和初等路

- 连通图

  在一个图中，如果任何两个节点之间，至少有一条链或路，则称该图为连通图，否则称为非连通图。如图6.1是连通图，图6.4为非连通图。

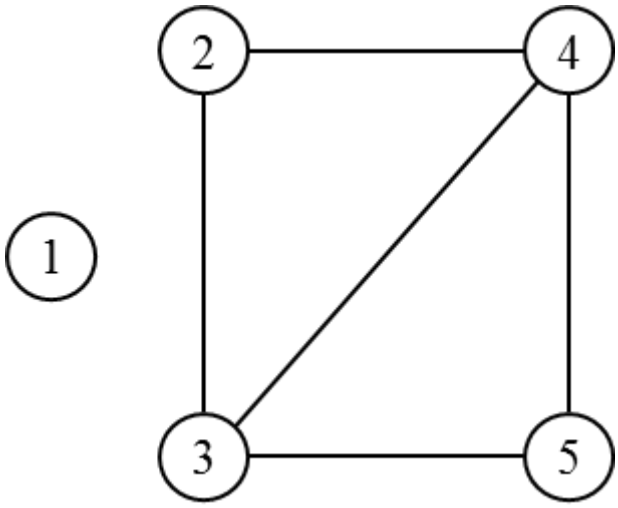
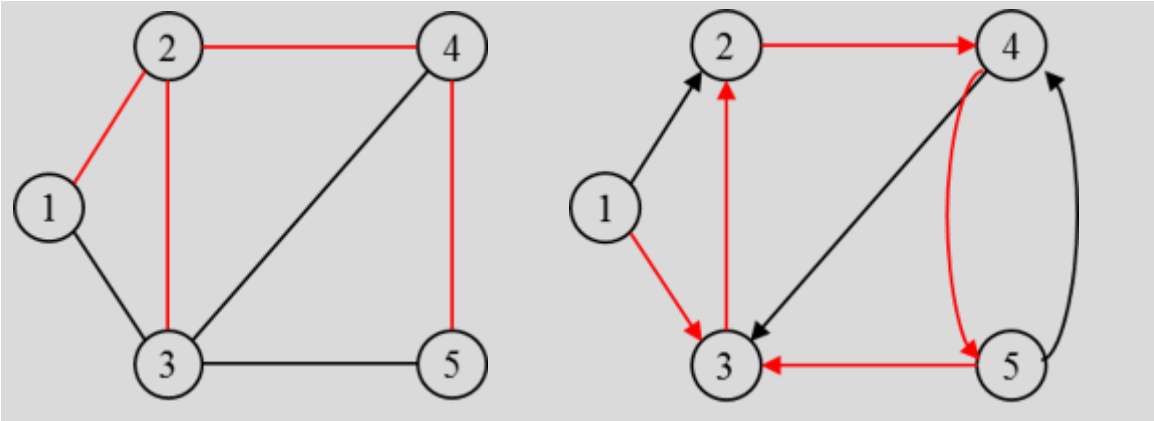


图6.4 连通图

- 树和生成树

  树是图论中最重要的概念之一。一棵树必然是无圈且连通的。一棵有m个节点的树，共有m-1条边。事实上，从树上每去掉一个节点，同时也一定会去掉一条边，最后当该树只剩下两个顶点时，也就只剩下一条边了（树的无圈连通性）。所以，树中边的数目比节点数目少1。反过来说，一个有m个节点m-1条边的连通图必是一棵树。

  生成树指的是包含图中所有节点的连通的树。“生成”一棵“树”只需要删除原网络中的所有边，再按一定规则依次添加上边，避免添加边后形成环。添加m-1条边后，恰好连接m个节点，此时得到生成树。图6.5中红色的边与节点连成的链即为原网络的一棵生成树。我们在后面的章节中将具体介绍计算一个网络中最小生成树的方法。



无向图的生成树　　有向图的生成树

图6.5 生成树

6.1.2 网络的表示方法

上一节，我们讨论的图仅涉及其拓扑结构，即节点与弧之间的连接关系，但这些并不足够，因为实际的网络问题仍需要对弧加以定量的描述。例如在交通网络图中，除了要描述各城市之间是否存在公路、铁路等运输线外，仍需要表达这些线路的长度、通行能力、运输成本等，这些数值统称为权。

为了方便在计算机上进行存储与计算，我们一般用矩阵表示一个网络。下面以有向图为例，介绍两种矩阵表示网络的方法。

- 节点-弧关联矩阵（Node-Arc Incidence Matrix）

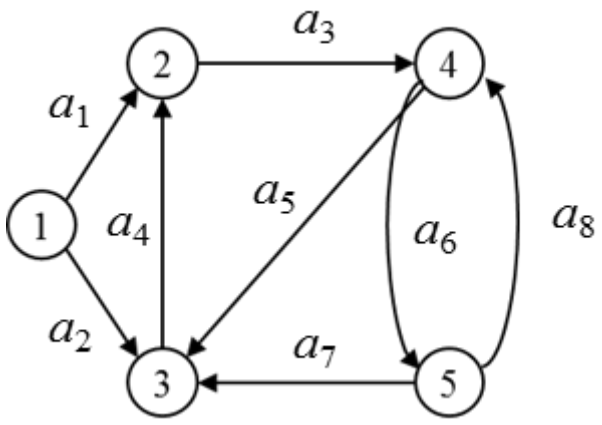
  节点-弧关联矩阵也可简称为关联矩阵，用来表示节点（n_i）和弧（a_j）的关联状态。每一行表示一个节点，每一列表示一条弧。关联矩阵中元素S_{ij}的定义如下：

$$s_{ij}=\begin{cases}1 & \text{若弧}a_j\text{自节点}n_i\text{射出}\\ -1 & \text{若弧}a_j\text{自节点}n_i\text{射入}\\ 0 & \text{若弧}a_j\text{自节点}n_i\text{不关联}\end{cases}$$

$$S_{ij}=\begin{cases}1, & \text{弧}a_j\text{自节点}n_i\text{射出}\\ -1, & \text{弧}a_j\text{自节点}n_i\text{射入}\\ 0, & \text{弧}a_j\text{与节点}n_i\text{不关联}\end{cases}$$

  以图6.1中的有向网络为例，我们来写它的关联矩阵（见图6.6）。该有向图有5个节点，8条边，所以关联矩阵是个5 × 8的矩阵。以弧a_1为例，该弧自节点1射出，节点2射入，所以s_{11}=1, s_{21}=-1。仔细观察，我们可以得到关联矩阵的以下特征：(1) 关联矩阵是个稀疏阵，在 $m \times n$个元素中，只有2m个非零元素，因此关联矩

阵不具有空间效率。(2) 每列都有两个非零元素，一个取值+1，一个取值-1，这反映了每条弧的两个端点及指向。



$$
\begin{array}{c}
 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 \\
n_1 & \left[\begin{array}{cccccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right. \\
n_2 & \begin{array}{cccccccc} -1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \end{array} \\
n_3 & \begin{array}{cccccccc} 0 & -1 & 0 & 1 & -1 & 0 & -1 & 0 \end{array} \\
n_4 & \begin{array}{cccccccc} 0 & 0 & -1 & 0 & 1 & 1 & 0 & -1 \end{array} \\
n_5 & \left.\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 \end{array}\right]
\end{array}
$$

图6.6 关联矩阵示意图

- 节点-节点邻接矩阵（Node-Node Adjacency Matrix）

节点-节点邻接矩阵，也可简称为邻接矩阵，表示各节点之间的连通状态。该矩阵每一行，每一列都对应一个节点。邻接矩阵元素h_{ij}的定义如下：

$$
h_{ij}=\begin{cases}1 & \text{若存在自节点}n_i\text{射向}n_j\text{的弧} \\ 0 & \text{若不存在自节点}n_i\text{射向}n_j\text{的弧}\end{cases}
$$

$$
h_{ij}=\begin{cases}1, & \text{存在自节点}n_i\text{射向}n_j\text{的弧} \\ 0, & \text{存在自节点}n_i\text{射向}n_j\text{的弧}\end{cases}
$$

图6.1中有向网络的邻接矩阵如下：

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}
\end{array}
$$

不难发现，邻接矩阵是$n \times n$个的方阵，含有m个非零元素。如果网络足够密集，邻接矩阵在空间上是高效的，因而常被用于网络算法中。此外，邻接矩阵也可用于存储路段费用和路段通行能力。

有了上述表征网络的方法，我们可以方便用计算机对网络流问题进行建模和求解。接下来，将介绍四种主要的网络流问题。

6.2 网络流问题建模

本节将介绍四大经典网络流模型：

- 最短路径问题 (shortest path problem)
- 最小生成树问题 (minimum spanning tree problems)
- 最大流问题 (maximum flow problem)
- 最小费用流问题 (minimum cost network flow problem)。

最小费用流是最一般的网络流问题，最短路径问题和最大流问题都是它的特殊形式。

6.2.1 最短路问题

最短路径问题是网络流模型中的一个最基本的问题。在生产实践，运输管理中，诸如工艺路线安排，管道线网铺设、设备更新等问题都可建模为最短路问题。

问题描述

下面看一个简单的例子。在1号小区有一个快递网点，快递小哥需要给位于8号小区的顾客派件，途中可能会经过若干个小区。为提高配送效率，快递小哥需要选择一条最快的路径到达8号小区。我们可以将上述问题抽象成如图6.7所示的网络图，弧上的权值表示小区间的距离，这种弧上有权值的图称为赋权图。该问题可以基于图论的方法在网络上求解。

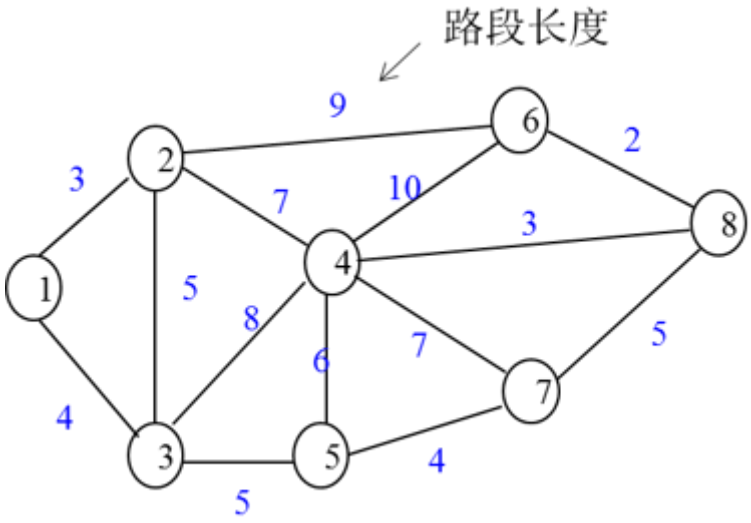


图6.7 快递配送示例网络

问题求解

在该例子中，我们要帮快递小哥找到1号小区到8号小区的最短路径，但是从1号小区无法直接到达8号小区，途中必须经过其它小区。其实，从1到8的最短路径，必定也是1号小区到该路径上某个中间节点的最短路径。

我们通过反证法来证明。在图6.8中，如果P是v_s到v_j的最短路径，而v_i是P中的一个点，那么在P上v_s到v_i的路径也是最短路径。如果Q是v_s到v_i的最短路径，P'是v_s沿着Q到v_i再沿P到v_j的路径，那么P'的权肯定小于P，这与P是最短路径相矛盾了。这证明了最短路径问题的最优性原理：最短路径的子路一定是最短路径。

根据最优性原理，找v_s到v_j的最短路径可以通过找v_s到v_i的最短路得到，而v_s到v_i的最短路又可以通过找其最短子路得到，由此不断地向前追踪，直至回到起点。由此看来，我们通过找v_s到v_j中间节点的最短路径，从而找到由起点v_s到终点v_j的最短路径。

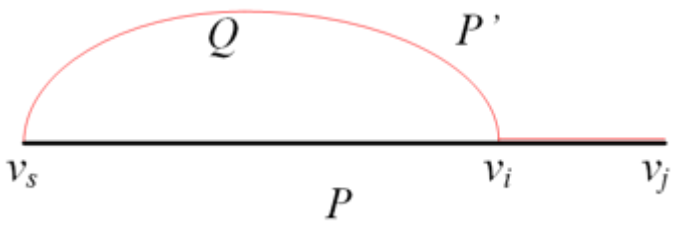


图6.8 最优性原理证明

Dijkstra算法就是根据这样一个思路来进行求解最短路径的。它是一个标号算法，将所有顶点分为两类，一类是有永久标号的顶点，另一类是临时标号节点。具体操作如下：

1. 初始化。算法初始化过程将起点标记为永久标号，标签设为0，其余节点为临时标号，标签为∞。

2. 迭代过程。按公式(6.1)更新与永久标号节点相邻节点的临时标号，并将临时标号中标签最小的节点变为永久标号（公式(6.2)）。需要注意，每次迭代有若干个临时标号得到更新，且只有一个编号能变成永久标号。

公式(6.1)

$$l_{k+1}(v_j) = min\{l_k(v_j), l_k(v^*) + W(v^*, v_j)\}$$

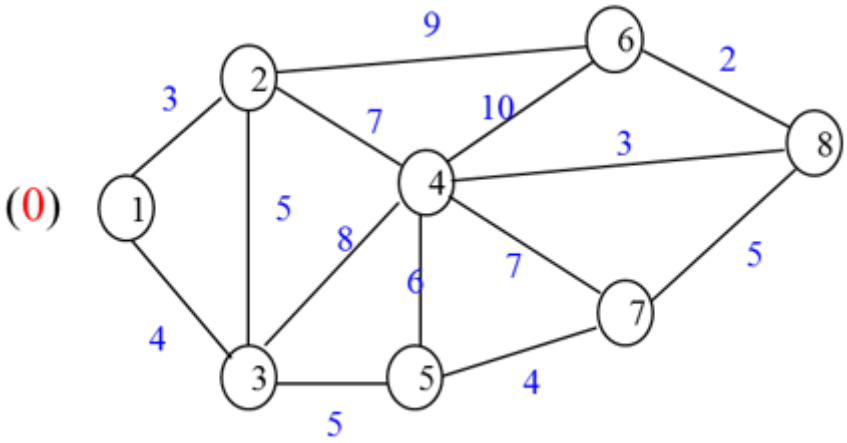
公式(6.2)

$$l_{k+1}(v^*) = min_{v_j \in \bar{A}}\{l_{k+1}(v_j)\}, \quad \bar{A} = A - \{V^*\}$$

其中，k表示第k次迭代次数，$l_k(v_j)$表示从起点开始到节点v_j的路径长度；初始化中，除起点外，其余节点的$l_k(v_j)$值都取∞；A表示所有节点的集合，$\bar{A}$表示临时标号节点的集合，v*表示永久标号节点。

例题. 用Dijkstra算法求解问题描述中的快递配送问题。

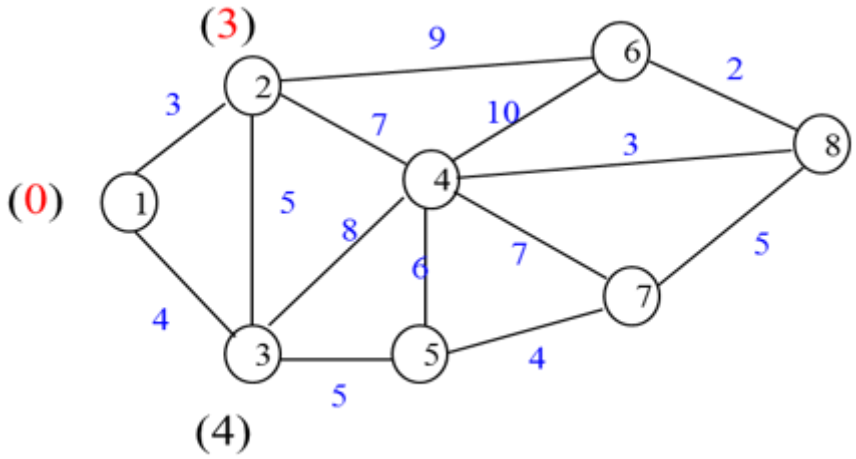
步骤**1**：初始化。将节点1设为永久标号，设为0，其余节点为临时标号节点，标签为∞。图中红色的数字表示永久标号的标签。



步骤**2**：更新与节点1相邻的节点2和节点3的标签。

$$l_2(v_2) = min\{\infty，\ 0 + 3\} = 3$$
$$l_2(v_3) = min\{\infty，\ 0 + 4\} = 4$$

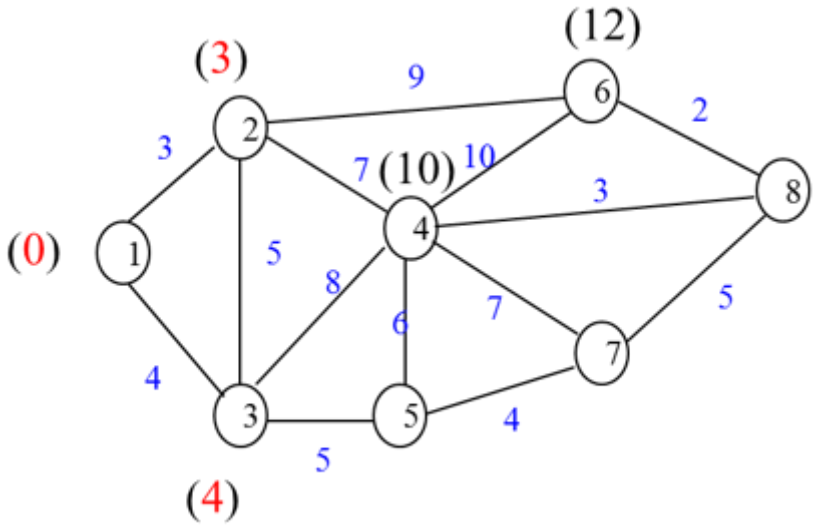
临时标号中，节点2的标签最小，将节点2变为永久标号。



步骤**3**：更新与节点2相邻的节点3、节点4和节点6的标签。

$$l_3(v_4) = min\{\infty，\ 3+7\} = 10$$
$$l_3(v_6) = min\{\infty，\ 3+9\} = 12$$
$$l_3(v_3) = min\{4，\ 3+5\} = 4$$

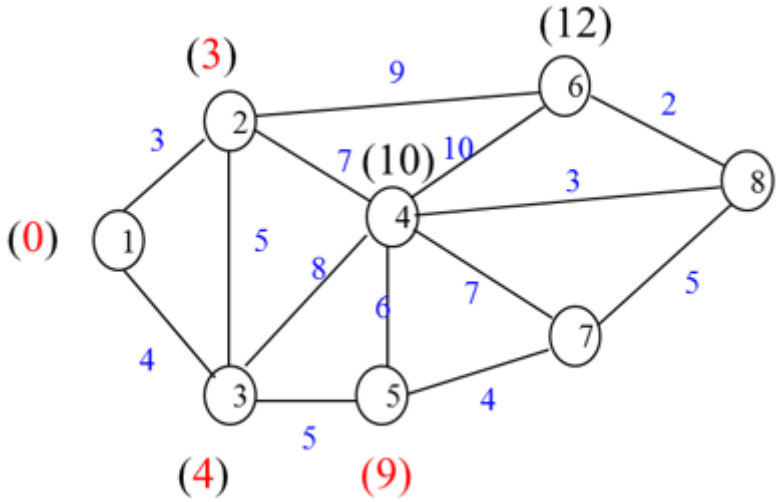
临时标号中，节点3的标签最小，将节点3变为永久标号。



步骤**4**：更新与节点3相邻的节点4和节点5的标签，保留节点6的临时标号。

$$l_4(v_4) = min\{10，\ 4+8\} = 10$$
$$l_4(v_5) = min\{12，\ 4+5\} = 9$$
$$l_4(v_6) = min\{12，\ \infty\} = 12$$

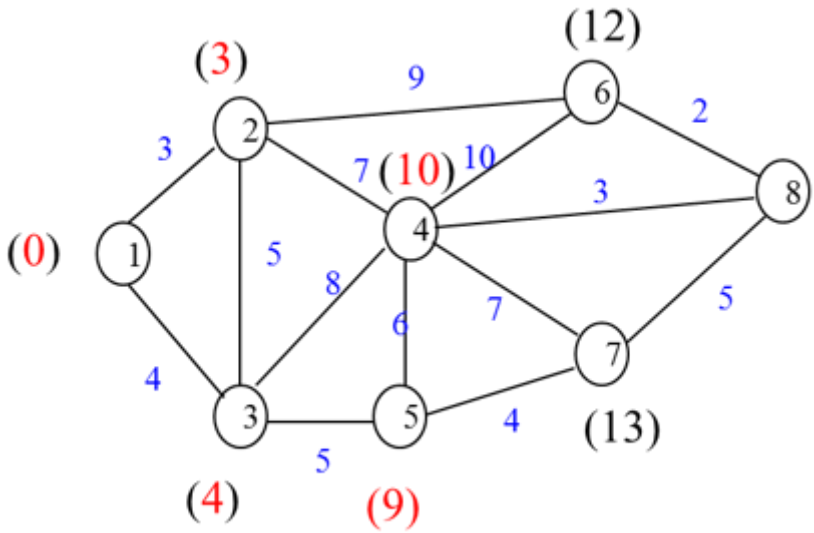
临时标号中，节点5的标签最小，将节点5变为永久标号。



步骤**5**：更新与节点5相邻的节点4和节点7的标签，保留节点6的临时标号。

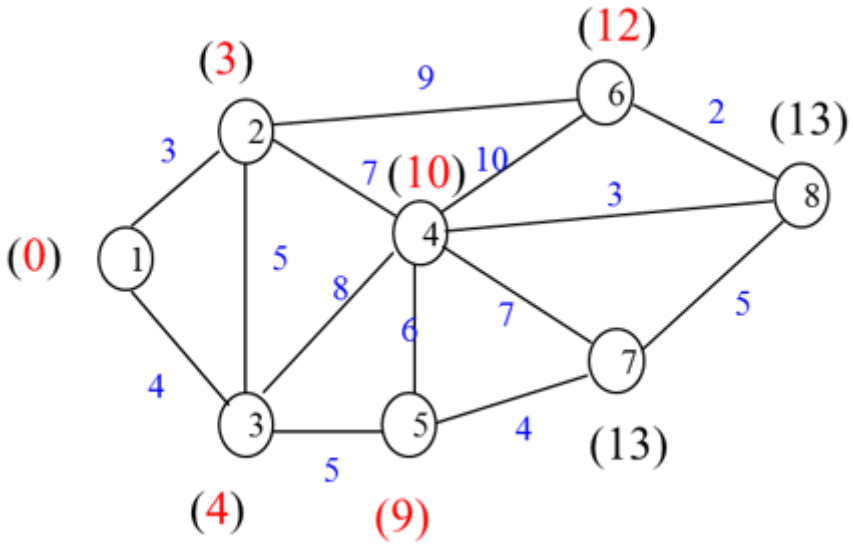
$$l_5(v_4) = min\{10，\ 9+6\} = 10$$
$$l_5(v_7) = min\{\infty，\ 9+4\} = 13$$
$$l_5(v_6) = min\{12，\ \infty\} = 12$$

临时标号中，节点4的标签最小，将节点4变为永久标号。



步骤**6**：更新与节点4相邻的节点6,7,8的标签。

$$l_6(v_6) = min\{12，\ 4+10\} = 12$$
$$l_6(v_7) = min\{13，\ 10+7\} = 13$$
$$l_6(v_8) = min\{\infty，\ 10+3\} = 13$$

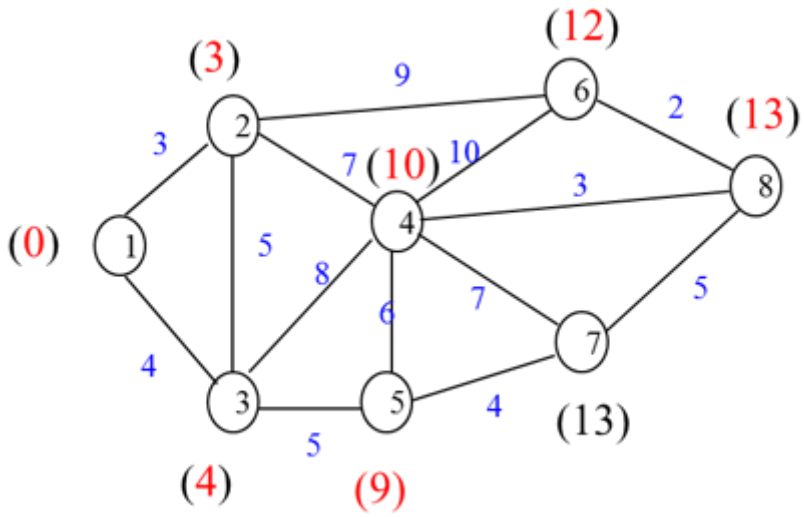


临时标号中，节点6的标签最小，将节点6变为永久标号。

步骤**7**：更新与节点12相邻的节点8的标签，保留节点7的临时标号。

$$l_7(v_8) = min\{13，\ 12 + 2\} = 13$$
$$l_7(v_7) = min\{13，\ \infty\} = 13$$

临时标号中，节点7和节点8标签相等，可任意选择其中一个将其变为永久标号。在此将节点8变为永久标号，可得到节点1到节点8的最短路径长度，为13.



步骤**8**：更新与节点8相邻的节点7的标签。

$$l_8(v_7) = min\{13，\ 13 + 5\} = 13$$

当前只剩下最后一个临时标号节点，将其变为永久标号，算法终止。

上述步骤也可以用以下运算表格体现，每一行表示每一次迭代过程，有星号标记的表示该标签已设为永久标号，无需进行标签更新。

	v1	v2	v3	v4	v5	v6	v7	v8
k=1	0^*	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
2		3^*	4	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
3			4^*	10	$+\infty$	12	$+\infty$	$+\infty$
4				10	9^*	12	$+\infty$	$+\infty$
5				10^*		12	13	$+\infty$
6						12^*	13	13
7							13	13^*
8							13^*	

到目前为止，我们找到了从小区1到小区8的最短路径长度为13，但是并不知道这条路径经过了哪些节点。下面，我们介绍用逆向追踪的方法来寻找最短径。从节点8开始，反向追踪找其紧前节点，需满足$l(v_j) = l(v_i) + w_{ij}$,由此得到由节点1到节点8的最短路径为$v_1v_2v_4v_8$.

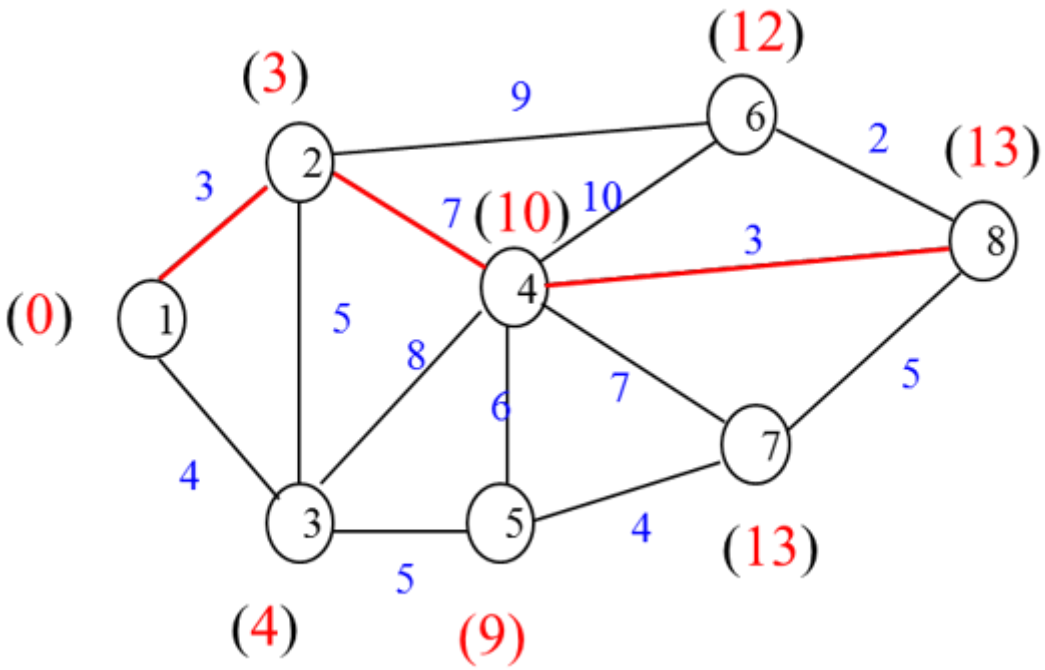


图6.9 示例网络最短路径

模型表达

求解节点1到节点m的最短路径问题不但可以转化成网络图，用专门的图论算法进行求解，还可以写成如下线性规划模型。

$$\text{Minmize} \quad \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{m} x_{ij} - \sum_{k=1}^{m} x_{ki} = b_i = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{if } i \neq 1 \text{ or } m \\ -1 & \text{if } i = m \end{cases}$$

$$x_{ij} = 0 \text{ or } 1 \qquad i, j = 1, \ldots, m.$$

$$Maximize \quad \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij}$$

$$s.t. \quad \sum_{j=1}^{m} x_{ij} - \sum_{k=1}^{m} x_{ki} = b_i = \begin{cases} 1, & if \quad i = 1 \\ 0, & if \quad i \neq 1 \quad i \neq m \\ -1, & if \quad i = m \end{cases}$$

$$x_{ij} = 0 \quad or \quad 1 \quad i, j \in \{1, \ldots, m\}$$

决策变量x_{ij}表示边(i, j)是否在最短路径上，是则取1，否则取0. 目标函数表示最小化这条路径上的费用。约束条件是针对节点的流量守恒约束：$\sum_{j=1}^{m} x_{ij}$ 表示节点i的流出量，$\sum_{k=1}^{m} x_{ki}$ 表示节点i的流入量。针对节点在网络中的不同位置，如起点、中间节点、终点，b_i表示节点i的净流量，分别取值为1, 0, -1. 其实，决策变量的0-1约束也可以被非负约束$x_{ij} \leq 0$代替，因为如果网络中存在最短路径的话，用单纯形法求解时，自然而然就能得到取值为0或1的整数解。此外，将变量设成非负约束还能够将原整数规划问题转换为线性规划问题，求解难度下降。

其它应用

路面更新问题：某新建公路设计年限为20年，使用若干年后，路面需更新（重铺路面）。把设计年限（20年）分成四个时期，每个时期为五年。假设每个时期内，各年的路面养护费及由于路面损坏而引起的附加行驶费用是不变的。又设路面更新是在某个时期的期末进行的，由于各个时期路面的损坏情况不同，故各个时期路面更新费用也不一样。试确定其使用期限20年内的路面更新计划使总费用最小。路面更新问题各期的具体费用见图6.10。

路面更新的年限	第1个五年	第2个五年	第3个五年	第4个五年
每年的养护费及附加费用(千元/km)	4	6.4	9.6	14.4
五年总费用(千元/km)	20	32	48	72

使用5年后更新	使用10年后更新	使用15年后更新
48千元/km	56千元/km	60千元/km

图6.10 路面更新问题描述

路面更新问题其实是求最小花费的路面更新方案，可以构建图网络，转换为最短路问题进行求解，见图6.11。

V_i表示“第i个时期末路面更新一次”的状态（i=1, 2, 3）
V_0表示道路刚建成之状态（这时路面是新的）
V_4表示道路到达设计年限的状态（这时不用再更新）
节点间连线上的数字表示各状态之间的路面更新费用、养护费及附加费用的总数
V_0到V_4的路径表示一个更新计划

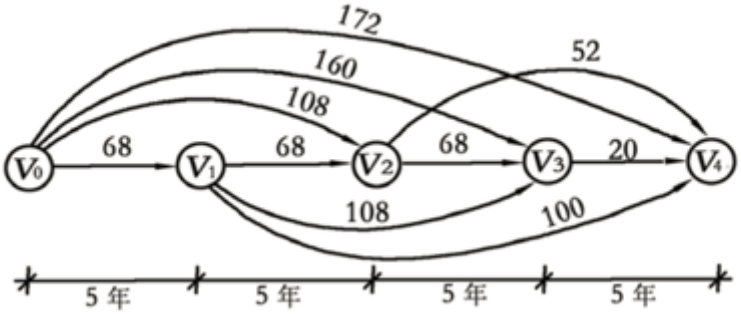


图6.11 路面更新问题网络图

一般而言，有以下几种最短路径问题：（1）在非负赋权图中，找一个节点到所有节点的最短路径；（2）在实数赋权有向图中，找一个节点到所有节点的最短路径；（3）找任意两个节点之间的最短路径。

本节中介绍的Dijkstra仅适用于非负权图的一对多的最短路径的求解。至于负权图，或求解多对多的最短路径问题，可以参考Floyd、A*、Bellman-Ford等算法。这些算法在网上及运筹学教材都能找到相应的资料（[4]-[6]）。

6.2.2 最小生成树问题

在图的基本概念这一小节中，我们介绍过生成树的基本概念了。对赋权图而言，它的生成树有n个节点，n-1条边。由于每条边有不同的权重，我们希望找到一棵各边权值之和最小的生成树，这样的问题被称最小生成树问题。

同样是考虑各边权值之和最小，那么最小生成树与最短路问题有什么区别呢？

1. 最小生成树能保证连通所有节点的路径之和最小，但不能保证任意两点之间是最短路径。最短路径是从一点出发，到达目的路径的最小值。
2. 一般而言，最小生成树针对无向图，最短路问题针对有向图。针对无向图的最短路问题很容易扩展到有向图上，但是在有向图上求一个给定树根的最小生成树（有向树）比求无向图中的最小生成树复杂得多。

相关应用

- 物理网络设计

  这类问题主要是设计一个网络，使得各网络的组成成分得以相连；或者是给位于不同地理位置的用户提供基础设施，使他们可以相互沟通。前者如在给某个地区的村庄布设道路，后者如架设一个通往各乡的电话线。这些问题的目标函数都要求建设成本最小，是最小生成树问题的一个典型应用。

- 最小生成树（**MST**）聚类

  聚类问题的核心就是将一组数据划分成不同的组，组内数据点间的距离要远小于组间距离。图6.12简单介绍了MST聚类的过程。图6.12(a)显示这是一个有27数据点的数据集。图6.12(b)是该数据集的最小生成树。现在要将该数据集分成四类，只需要删除该最小生成树上的三条边。我们在这里选择距离最长的三条边，如图6.12 (c)所示，这样才能保证各簇团内的点距离最小，且小于簇团间的点。

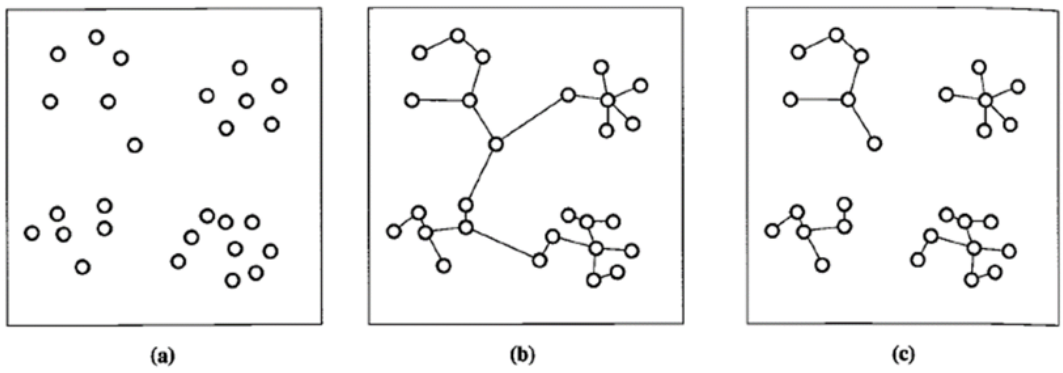


图6.12 最小生成树聚类

问题求解

我们通过具体算例来介绍求解最小生成树问题的算法。

图6.13所示的赋权连通无向图G中各顶点表示某个地区的8个乡，边上的权表示乡之间的距离。现欲架设一个通往各乡的电话线网，问应该如何架设电话线网而使其总长度最短？

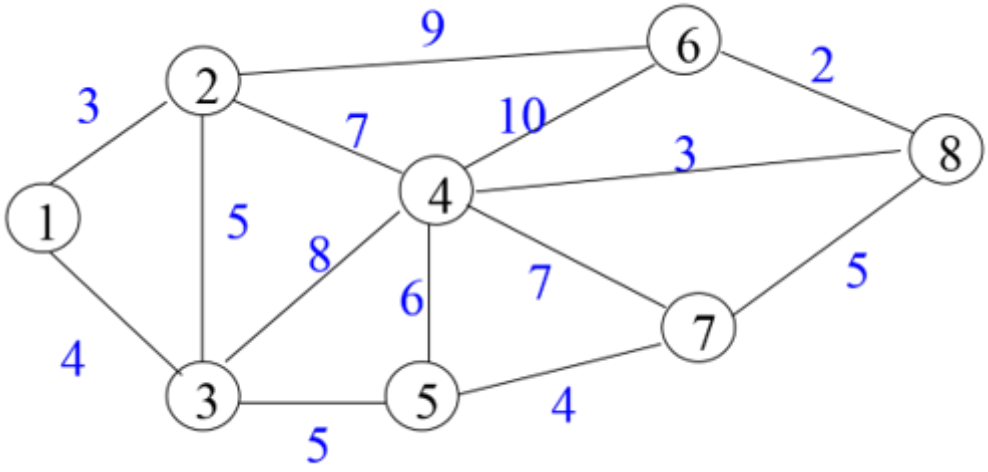


图6.13 最小生成树算例网络

上述问题实际上就是找G中的一颗生成树，使得树各边权和最小。我们分别用破圈法和避圈法进行求解。

方法一：破圈法

基本思想：因为无回路且连通是树的特性。为此，我们保证各点之间的连通性，每次在图中任取一个回路，删去权最大的边，直至图中无回路为止。这样剩下的边都是权值最小的，得到的生成树就是最小生成树。

具体操作：该算例的求解步骤从权重最大的边（10）开始删除，然后再删除剩余边中权重最大的（9），再依次删除权重为8,7,6,5的边，直到图中不存在回路为止。读者可能感觉到这并不是先选回路再选择权重大的边，也没有体现选择回路时的随机性。不过读者可以自己练习一下，先随机取一个回路，再删除其中权最大的边，得到的结果是一样的。最终求得的最小生成树的权值为3+4+5+4+5+3+2=26。

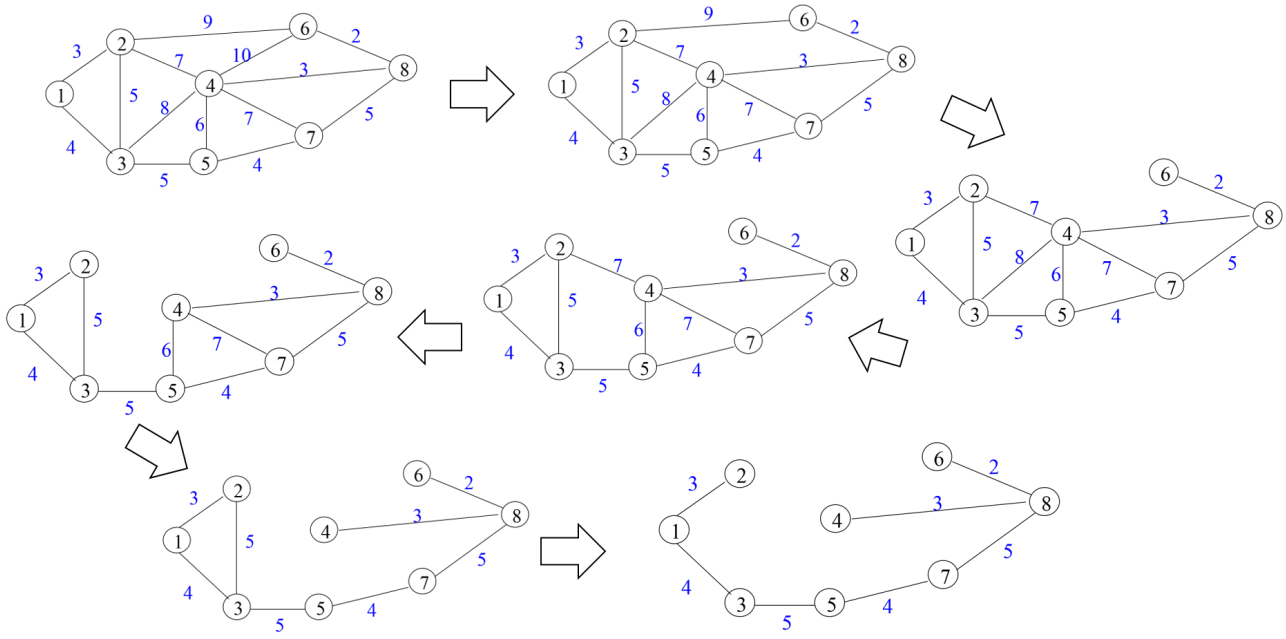


图6.14 破圈法具体步骤

方法二：避圈法

基本思想：生成树无圈且边数为n-1。该算法根据“在无圈的条件下优先选取权小的边”这一原则，从图的m条边中逐个挑选出n-1条边。注意：新加入的边不能与已挑选的边组成圈。

具体操作：

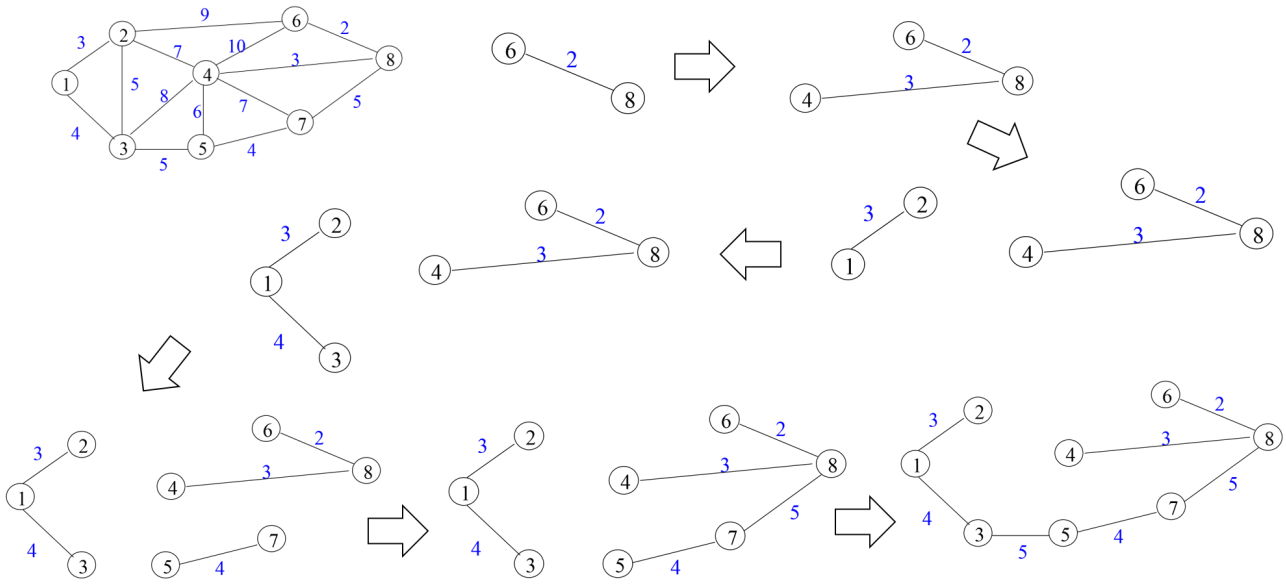


图6.15 避圈法具体步骤

最终求得的最小生成树的权值为3+4+5+4+5+3+2=26。两种方法求得的最小生成树和权值都一样。注意，最小生成树有可能不唯一，但是树的权值一定唯一。

6.2.3 最大流问题

在此之前，我们研究的网络流问题都仅涉及网络的节点，边及边上的权值，在本节我们将引入流的概念。流，网络上承载的实体。不同的物理网络，承载的流也各不相同。在城市路网中，有可能是小汽车，公交车，自行车等；在通信网络中，流是传递的信息；在电网中，流是传输的电量；在企业的生产网中，流是传送带上的产品。不管这个网络上运送的是什么样的流，都需要考虑这样一个问题：这个网络能承载的最大流量是多少？这就是我们本节要讨论的最大流问题。

问题描述

我们讨论运输网络中的最大流问题。在图6.16中，A1和A2处分别有物资12吨和24吨，B1和B2处分别需要物资16吨和20吨， F1、F2和F3为转运点，边上的数字为该运输线路运送该物资所允许的最大输送量。如何调运物资，使得A1和A2处有最多的物资输送到B1和B2处？

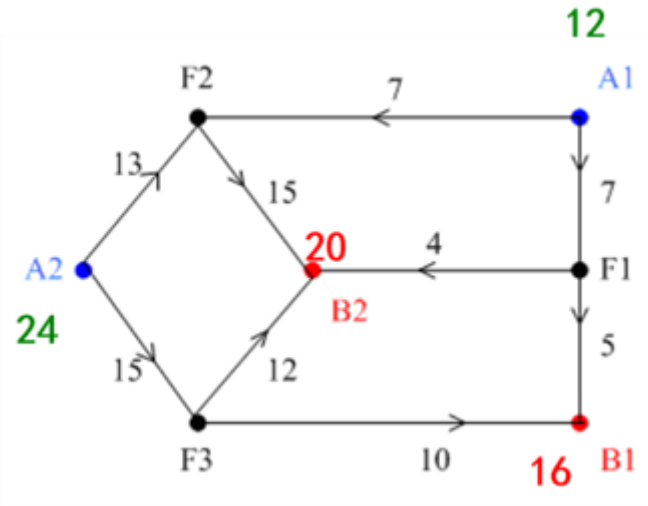


图6.16 运输网络

在图6.16中，存在三种类型的节点：运输流的发点（源）——A1和A2，收点（汇）——B1和B2，中间节点——F1、F2和F3。这个例子中，边上的数字不表示运输成本，仅表示边的容量。所以，在这个运输网络中，我们不要考虑运输成本，只需要保证每条边上运输的物资不能超过它能承载的最大容量，并考虑如何将更多的产品从发点运送至收点。

下面我们将介绍最大流问题中的一些概念。

流：设$f(e)$为边e的流值，$f^+(v)$为以v为起点的所有有向边流出量的和；$f^-(v)$为以v为终点的所有有向边流入量的和。如果$f(e)$满足以下两个条件：

- 容量约束条件，即有$0 \le f(e) \le u(e), \forall e \in E$ ；
- 中间节点流量守恒，即$f^+(v)=f^-(v), \forall v \in I$，中间节点的流出量=流入量；

称f为网络上的一个流。

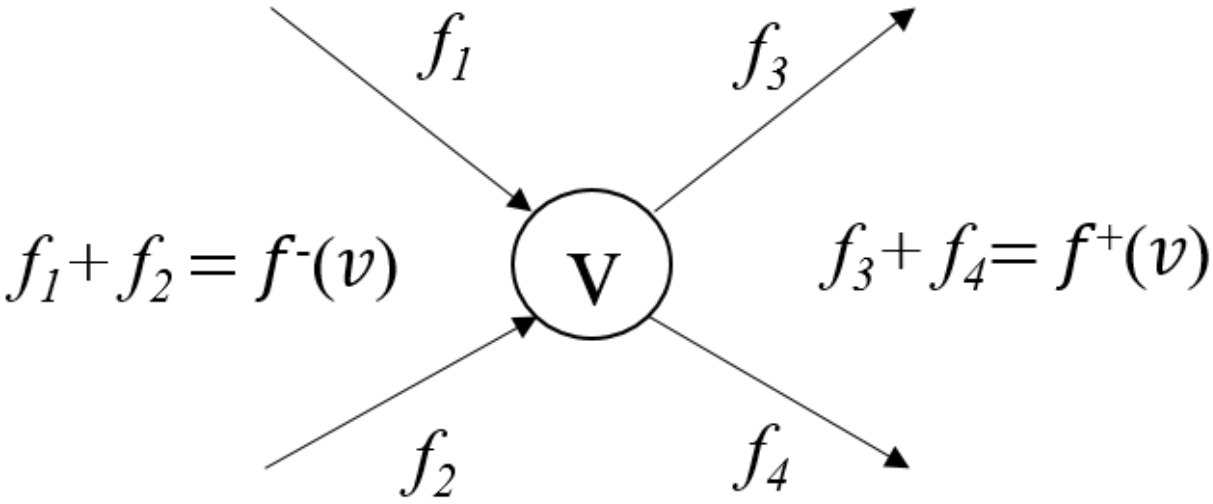


图6.17 流量守恒约束

单源单汇运输网络：在本节中我们讨论的运输网络有且仅有一个源一个汇，称为单源单汇网络。图6.16所示的运输网络是多源多汇网络，我们将图中的节点按源，中间节点，汇的顺序重新排列，将其转换成图6.18所示的单源单汇网络，并给出了一种可能的流值。边上两个参数的含义分别表示边的容量和流值（$u(e)$，$f(e)$）（后面若无特别说明，都为此含义），整个网络的流Val $f=f^+(x)=f^-(y)$。当前运输方案下的网络流为35，即从出发点x1输出物资12吨，从出发点x2输出物资24吨。当前的方案是该网络可以承载的最大流吗？这就是本节我们要关注的问题。

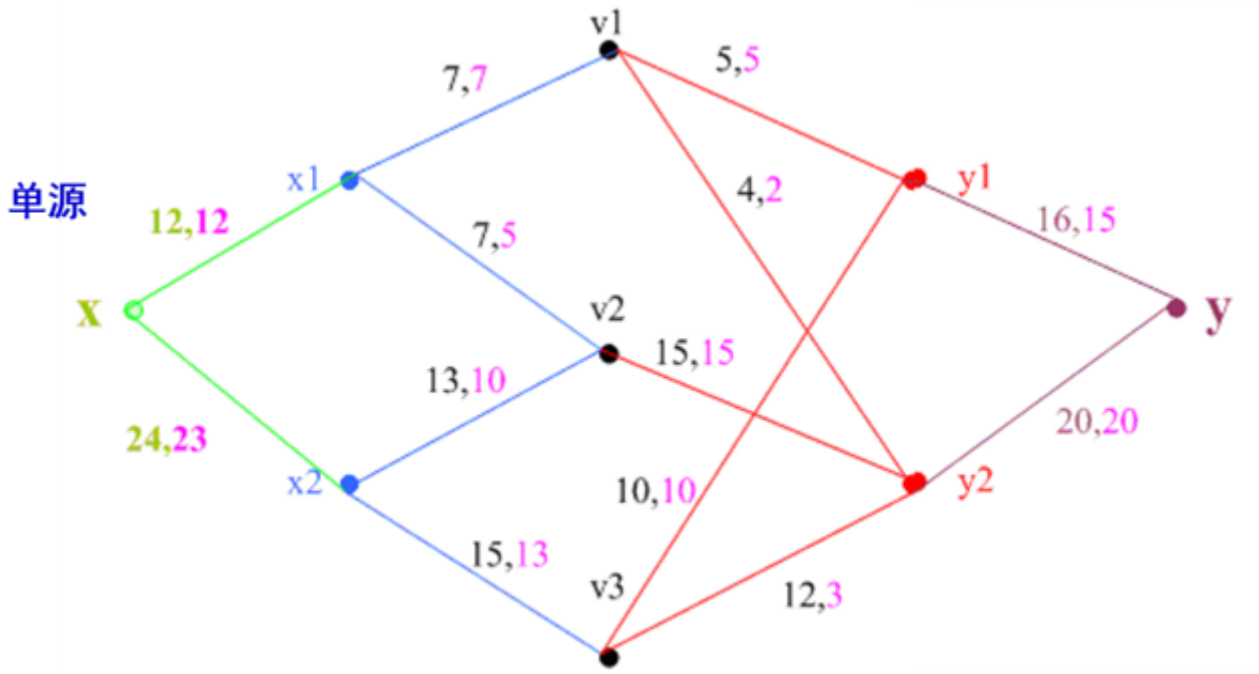


图6.18 单源单汇运输网络

问题求解

本节将介绍标号算法求解最大流问题，在此之前先介绍增广链这一概念。

首先，什么是链？链面向的是基本图，并不需要方向。增广链还是一条初等链，即链中不可以有重复的节点，其实这与路径的概念有些相似，只不过路径需要朝着同样的方向前进。所以，路径一定是初等链，但初等链不一定是路径。图6.19给出了一条由x到y的初等链，由于v5-v4, v4-v3的方向与x-y的方向（也就是前进方向）相反，所以这条初等链不是路径。我们将与前进方向相反的有向边称为后向边，与前进方向相同的有向边称为前向边。

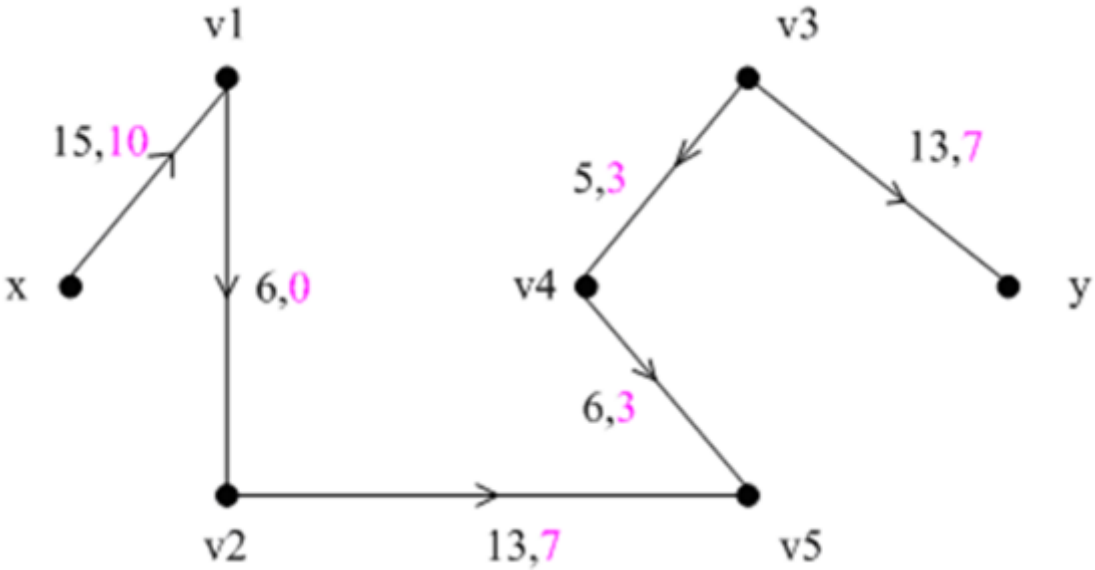


图6.19 一条初等链

对于前向边，我们关注这条边是不是饱和边，也就是说前向边上的流量是否等于边的容量。如果不相等，则说明该边非饱和，还可以增加流量直至饱和。对于后向边，我们关心这条边是不是零流边，即后向边上的流量是否等于零。对于非零流的后向边，我们希望减少该边上的流量，最好能使流量为0。为什么呢？其实可以这么理解，后向边与链的前进方向相反，也就是“逆行”了，属于“无效”运输，所以我们要尽可能减少后向边的流量，让更多的流量集中在前向边上，保证流量的有效运输。

因此，我们需要找到满足以下条件的初等链：前向边有非饱和边，后向边中有非零流边，通过调整链上的流量，可提高流值，这就是刚开始提到的增广链。由此可见，图6.19中的初等链是一条增广链。

具体的调整方法如下：

设Q为一条初等链，f是当前网络上的流，对 $e \in Q$，令：

$$l(e)=\begin{cases}C(e)-f(e), & \text{当}e\text{是}Q\text{的前向边}\\ f(e), & \text{当}e\text{是}Q\text{的后向边}\end{cases}$$

$$l(Q)=\min\{l(e)|e\in Q\}$$

$$l(e)=\begin{cases}u(e)-f(e) & e\in Q,\text{前向边}\\ f(e) & e\in Q,\text{后向边}\end{cases}$$

$$l(Q)=min\{l(e)|e\in Q\}$$

$l(Q)$即为该链上的流量调整量，该网络上的新流 $\tilde{f}$可以通过以下方法得到：

$$\hat{f}(e)=\begin{cases}f(e)+l(Q), & \text{当}e\text{是}Q\text{的前向边}\\ f(e)-l(Q), & \text{当}e\text{是}Q\text{的后向边}\\ f(e), & \text{其他}\end{cases}$$

$$\tilde{f}=\begin{cases}f(e)+\delta & e\in Q,\text{前向边}\\ f(e)-\delta & e\in Q,\text{后向边}\\ f(e) & e\notin Q\end{cases}$$

此时，有$Val\tilde{f}=Valf+l(Q)$,称$\tilde{f}$为f基于Q的修改流。

所以，要判断当前流是否为最大流，只需要判断当前网络中是否存在增广链。不存在增广链即为最大流。到目前为止，求一个网络的最大流问题转换为寻找网络中的增广链。

下面介绍标号法寻找增广链的过程。

1. 发点x有$l(x)=+\infty$，标号（-，$+\infty$）

2. 选择一个已标号的顶点u，对其所有未标号的邻接点v，按照下述规则标号：

(1) 若边$(u,v)\in E$，且$f(u,v)<u(u,v)$时，令

$$l(v)=\min\{l(u),\ u(u,v)-f(u,v)\}$$ **前向边**

给v点标号（+u，$l(v)$）

(2) 若边$(v,u)\in E$，且$f(v,u)>0$时，令

$$l(v)=\min\{l(u),f(v,u)\}$$ **后向边**

给v点标号（-u，$l(v)$）

3. 重复上述过程，直至收点被标号或不再有顶点可标号为止。

标号中的第一个参数u表示在该增广链上与之相邻的紧前节点，该参数的正负号仅表示方向；第二个参数$l(v)$表示允许的调整量，计算方法为min{紧前节点的标记值,当前节点调整量}。调整量的计算方法因前向边和后向边而有所不同。

我们得到最大流算法的具体步骤如下：

1. 初始流f（如零流）
2. 用标号法寻找f的增广链Q
3. 增广链上的流量调整过程
4. 重复上述两过程，直至不存在增广链为止。（即终点得不到标号）

例题：求图6.20示例网络中的最大流，边上的参数为（$u(e)$，$f(e)$），当前网络流f为17。

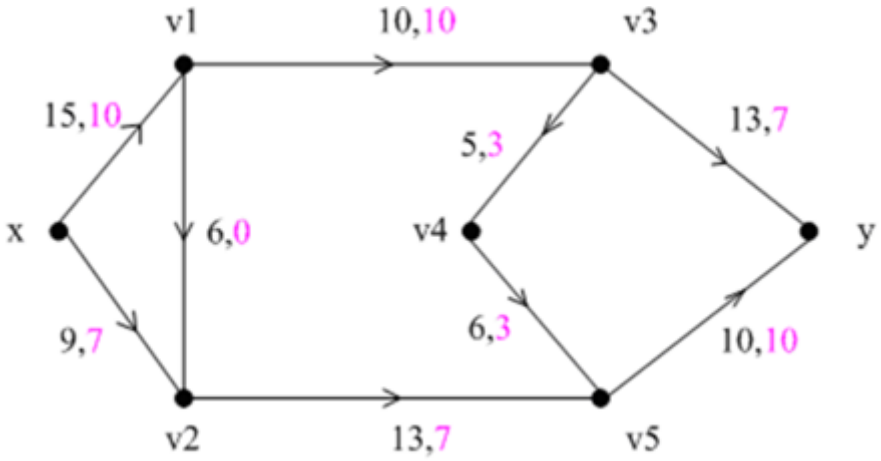


图6.20 最大流问题示例网络

步骤**1**：由于当前网络已经给出了初始流，下面直接用标号法寻找f的增广链。

从节点x出发，按照前向边和后向边的编号更新规则，依次对节点进行标号。各节点标号结果及所求的增广链如图6.21所示，括号左边的参数表示当前节点相邻的紧前节点，该参数的正负号仅表示方向；括号右边的参数表示当前节点连接紧前节点的边上允许的调整量。

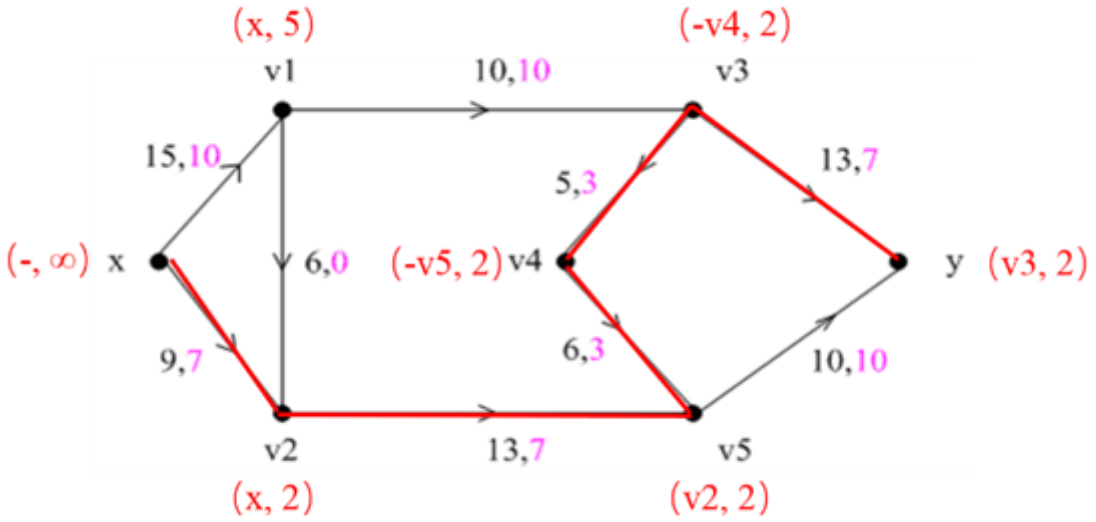


图6.21 第一次标号结果

步骤**2**：增广链上的流量调整

我们找到这样一条增广链x-v2-v5-v4-v3-y，整条链上允许调整的最小流量为2单位。后向边(v4, v3), (v4, v3)减少2个单位流量，其余边为前向边，增加2个单位的流量。调整后的网络流如图6.22所示，流值为19。

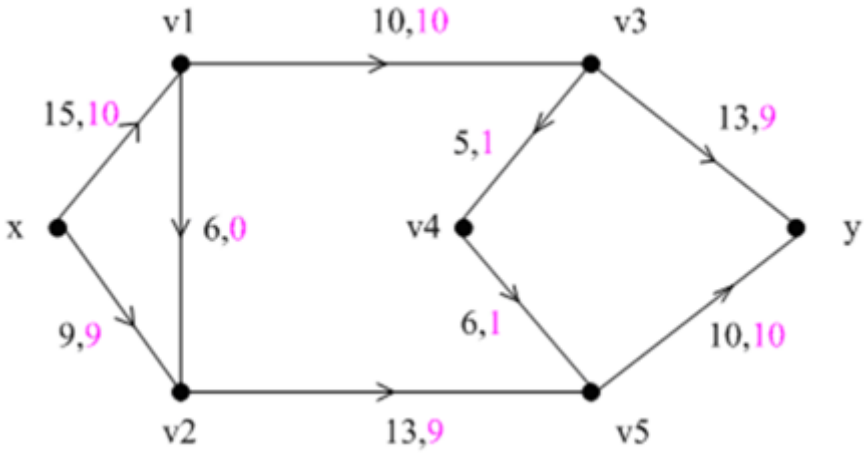


图6.22 增广链上的流量调整结果

步骤**3**：继续用标号法寻找*f*的增广链，结果如图6.23所示。

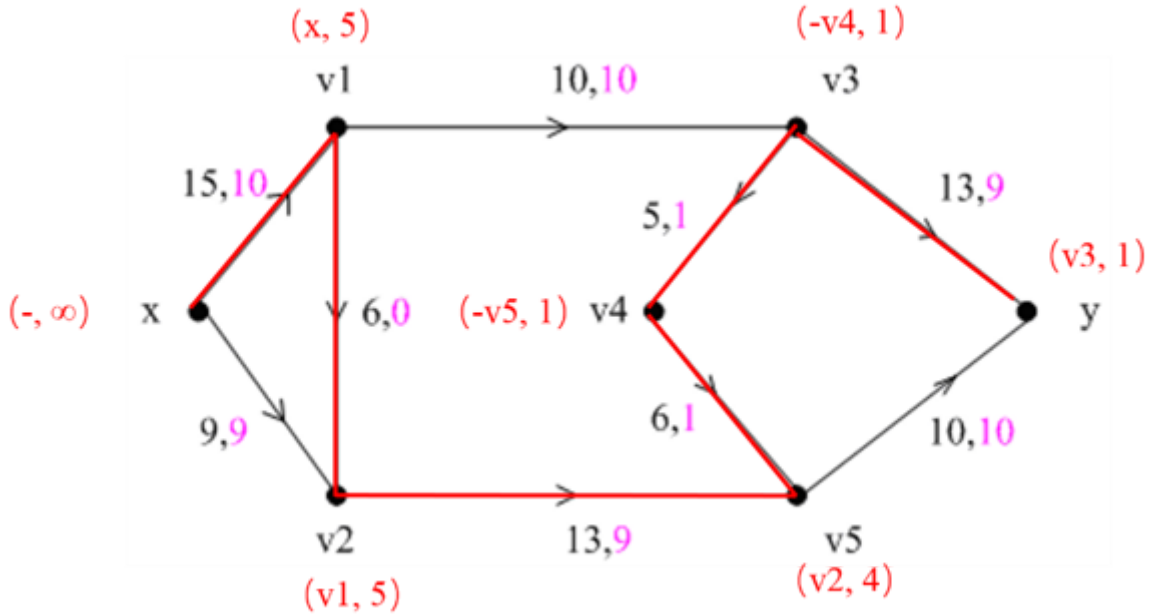


图6.23 第二次标号结果

步骤**4**：增广链上的流量调整

增广链x-v1-v2-v5-v4-v3-y 上允许调整的最小流量为1个单位，调整后的网络流及节点标号如图6.24所示。

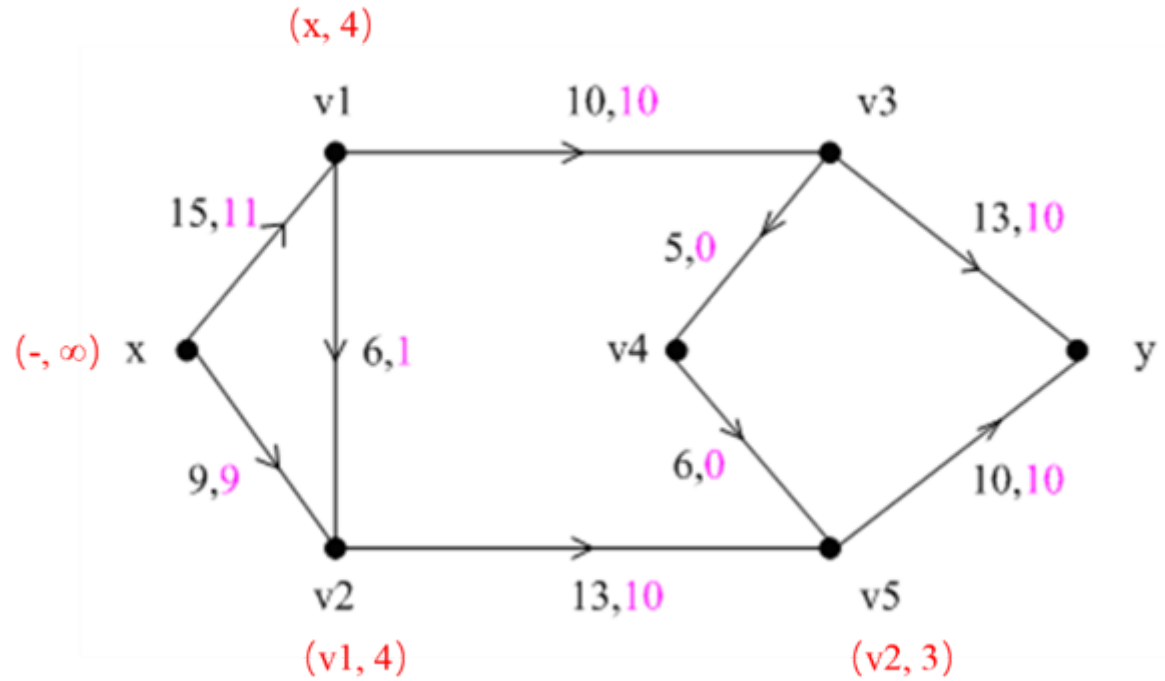


图6.24 第三次标号结果

由于收点y无法标号，既不存在增广链，所以当前流为最大流，流值为20。

模型表达

除了应用上述的标号算法，我们还可以将最大流问题写成以下的线性规划模型，用单纯形法或者优化求解器进行求解。

$$\text{Maxmize} \quad f$$

$$\text{subject to} \quad \sum_{j=1}^{m} x_{ij} - \sum_{k=1}^{m} x_{ki} = b_i = \begin{cases} f & \text{if } i = 1 \\ 0 & \text{if } i \neq 1 \text{ or } m \\ -f & \text{if } i = m \end{cases}$$

$$0 \leq x_{ij} \leq u_{ij} \qquad i, j = 1, \ldots, m$$

$$Maximize \quad f$$

$$s.t. \quad \sum_{j=1}^{m} x_{ij} - \sum_{k=1}^{m} x_{ki} = b_i = \begin{cases} f, & if \quad i = 1 \\ 0, & if \quad i \neq 1 \quad i \neq m \\ -f, & if \quad i = m \end{cases}$$

$$0 \leq x_{ij} \leq u_{ij} \quad i, j \in \{1, \ldots, m\}$$

变量f表示网络N上的流值，即为所求的最大流，变量x_{ij}表示边(i, j)上的流值大小，为b_i节点i的净流量。第一个约束条件是针对源，中间节点，汇的流量守恒约束；第二个约束条件是容量约束，u_{ij}为边(i, j)上的容量。

最大流最小割定理

在本节的最后，我们补充介绍一个重要的定理——最大流最小割定理。

割，是有向边的集合。对运输网络，如果把发点x所在的集合设为S，收点y所在的集合设为$\bar{S}$，则由集合S指向集合$\bar{S}$的边称为一个割。主要注意，网络中的节点不是在集合S，就在集合$\bar{S}$中。割边的容量之和称为割容量。如果把割集从网络中移除，余下的图不一定能分成两部分，但一定能把从x到y的路径切断，此时就不存在x到y的流。

最大流最小割定理：网络中从**x**到**y**的最大流等于分离**x**和**y**的最小割的容量。

最大流最小割定理直观上并不难理解。割集可以看成网络中的瓶颈，那最小割容量的割集就是整个网络中“最窄”的瓶颈，是影响网络流增加的关键因素。故网络中存在最大流，必等于该网络的最小割容量。

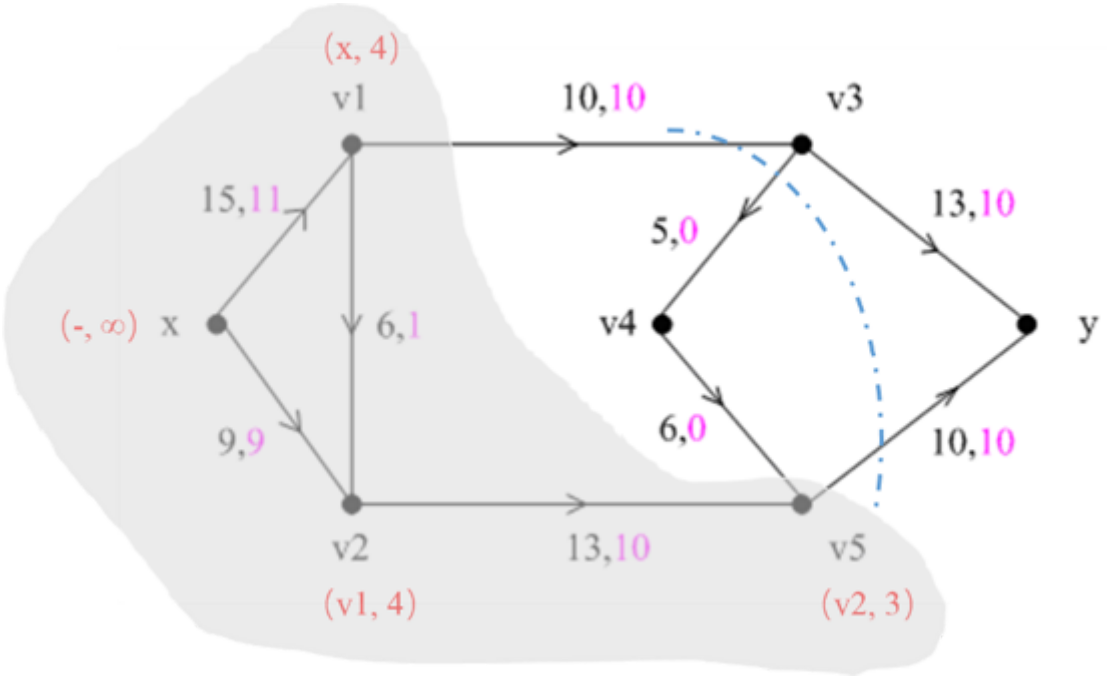


图6.25 割集

6.2.4 最小费用流问题

问题描述

对运输企业而言，控制运输成本的方法主要有两个：（1）单次运输过程中运送尽可能多的货物（这也是为什么货车经常超载的原因）；（2）选择运输费用最小的路径，如不饶远路，不走高速等。这涉及到网络流中的一个经典问题：最小费用流问题。

首先，什么是最小费用流问题？我们仍以运输网络为例进行介绍。从字面上理解，最小费用流问题是要找到使得运输费用最小的运输方案。当然了，费用是与运量相关的。运输的货物多，费用自然高了；运输的货物少，费用自然就低。那么如何表示运量跟费用呢？

设f是网络上一个流（即一种运输方案），$Val\ f=\lambda$（即运量），编号为e的边上的权值W(e)表示在e边上运送一个单位的流量所花费的费用，$f(e)$表示在e边上运送的货物量。

那么整个运输方案的费用可以这么计算：

$$W(f)=\sum_{e\in E}W(e)f(e)$$

$W(f)$也就是流f的费用，E为网络中边的集合。

最小费用流：当运量为λ的情况下，使得$W(f)$最小的*$\mathbf{f}$称为$\mathbf{N}$上一个流值为λ**的最小费用流。

最小费用最大流：若λ为网络上最大流的流值，则称使得$W(f)$ 最小的*$\mathbf{f}$**为最小代价的最大流。

问题求解

求解最小费用流问题，就是在运量一定的前提下要找到一个运输费用最小的运输方案。这个问题有两个目标，一是要保证运输的流值，二是要确保费用最小。目前的算法无法一步到位求得最小费用流，只能分步实现目标。

这么看来，在网络中求流值为λ最小费用流有以下两种思路：

（1） 给定流值λ，寻找最小费用流。在网络中任取一个流值为λ的流f，通过不断调整，降低费用，最终求得最小费流。

这一思路要解决的关键问题是如何调整f，使得费用下降，流值不变？

（2） 给定最小费用流，提高流值λ。在网络中任取一个流值小于λ的最小费用流f（例如，f为零流），通过不断调整，将流值提高到λ。

这一思路要解决的关键问题是如何调整f，使得流值提高，费用仍是最小？

在最大流问题这一节中介绍了如何在增广链上提高流值，那么如何在保证流值不变的情况下，降低费用呢？能否找到这样一条链，在这条链上调整流量，流值不会发生变化，但是费用可以降低。为此，这里引入增流网络N_f这一概念。

增流网络N_f与原网络N有相同的节点。N_f上两节点的边(u,v)既可以是前向边（可增加流值），也可以是后向边（可降低流值），也可以两者兼有，取决于N上$f(e)$与$u(e)$的关系。增流网络N_f边上的参数仅有边的容量及权值，记为(u'(e)，W'(e))。

① 如果N有$f(e)<u(e)$，即非饱和边，那么N_f对应的边上既有前向边，也有后向边。前向边的容量$u'(e)=u(e)-f(e)$，$W'(e)=W(e)$，表示可以该边上可以增加的流值；后向边的$u'(e)=f(e)$，$W'(e)=-W(e)$，表示该边上可降低的流值。

② 如果N有$f(e)=u(e)$，即饱和边，那么N_f对应的边上仅有后向边。饱和边是无法增加流值的，但是可以反向降低流值。同理有$u'(e)=f(e)$，$W'(e)=-W(e)$

③ 如果N有$f(e)=0$，即零流边，那么N_f对应的边上仅有前向边。零流边可以增加流值，但无法降低流值。同理有$u'(e)=u(e)-f(e)=u(e)$，$W'(e)=W(e)$。

下面我们给出一个构造增流网络N_f的例子，见图6.26，边上的参数从左往右依次是（u(e)，f(e)，W(e)）。在原网络N中，（x, $v1$）和（$v1$, $v2$）是非饱和边，所以增流网络N_f中既有前向边（x, $v1$）和（$v1$, $v2$），也有后向边（$v1$, x）和（$v2$, $v1$）。（x, $v2$）和（$v2$, y）是饱和边，所以N_f中仅有后向边（$v2$, x）和（y, $v2$）。因（$v1$, y）是零流边，所以N_f中仅有前向边（$v1$, y）。

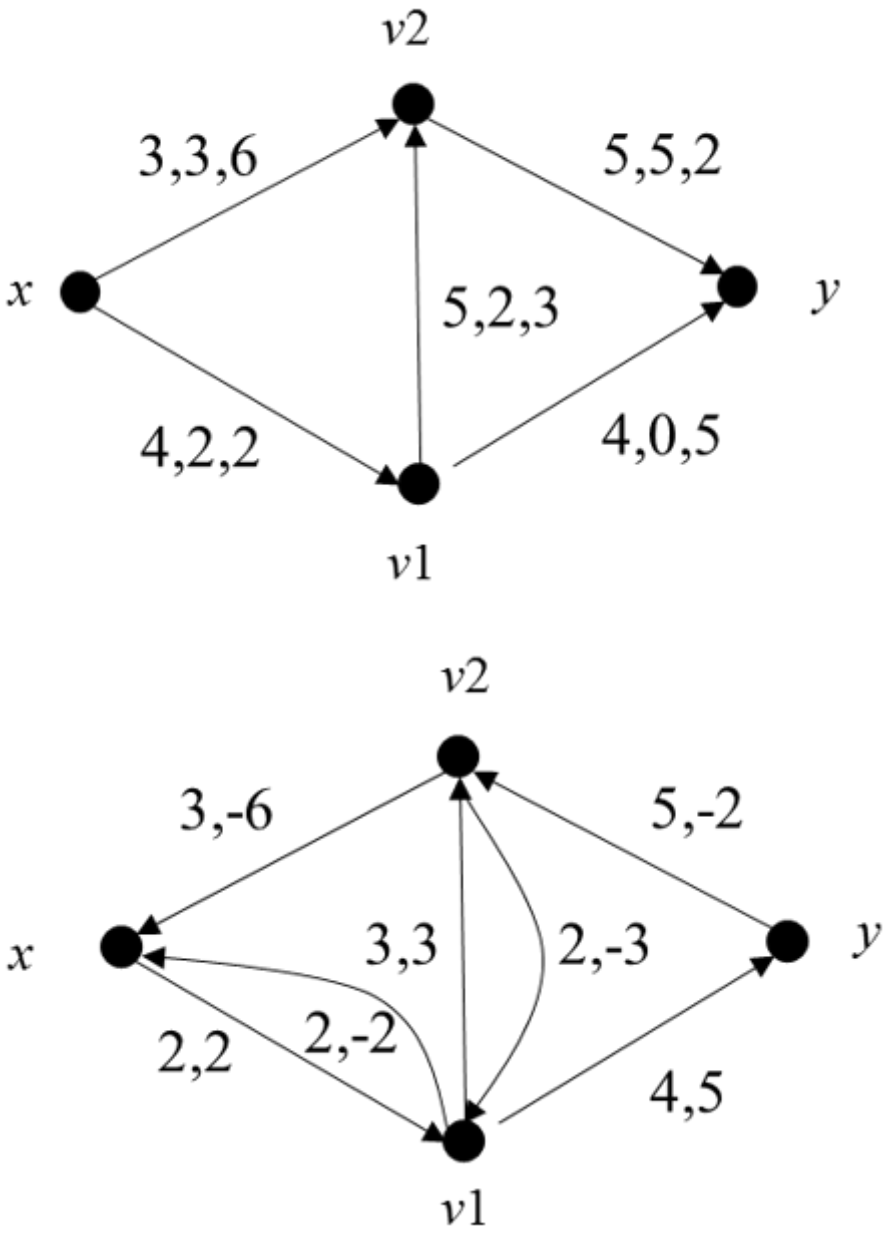


图6.26 构造增流网络示例

接下来我们介绍如何在增流网络N_f求得最小费用流。

针对上面提出的两种思路，我们分别给出两种求流值为λ最小费用流的算法。

算法一：给定流值λ，寻找最小费用流

算法步骤如下：

① 在N中任取一个流值为λ的流f

② 作增流网络N_f

③ 判断N_f中是否存在负回路Q（回路权值为负）？

- 若存在Q，则按回路容量调整f，调整量δ是Q中容量的最小值，即
  δ=min{ C'(e) | e为Q中有向边}
  在N上得到调整后的流$\tilde{f}$

$$\tilde{f}=\begin{cases} f(e)+\delta & e\in Q,\text{前向边} \\ f(e)-\delta & e\in Q,\ \text{后向边} \\ f(e) & e\notin Q \end{cases}$$

$$\tilde{f}=\begin{cases} f(e)+\delta & e\in Q,\text{前向边} \\ f(e)-\delta & e\in Q,\text{后向边} \\ f(e) & e\notin Q \end{cases}$$

因为在回路上调整流量，所以流值不发生改变。

修改当前流的费用：

$$W(\tilde{f})=W(f)+\delta\sum_{e\in Q}W'(e)$$

因为Q为负回路，所以有 $\sum_{e\in Q}W'(e)<0$,当前流的费用降低。返回步骤②。

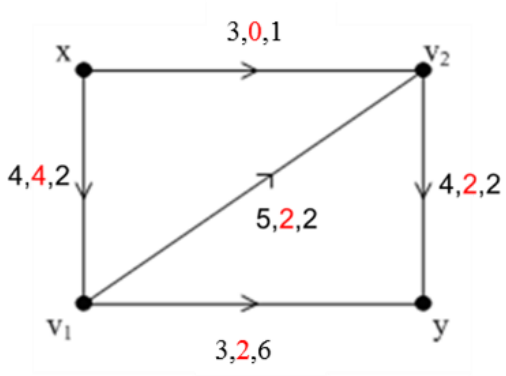
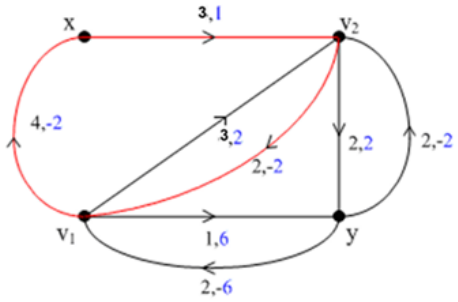
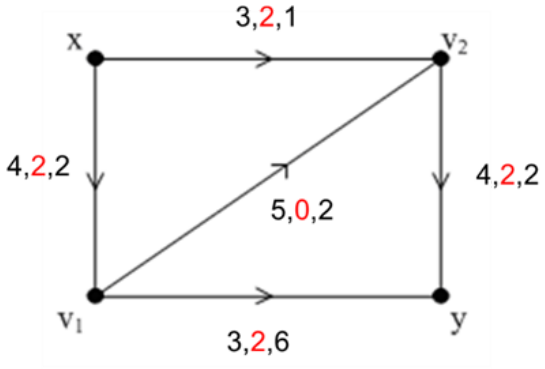
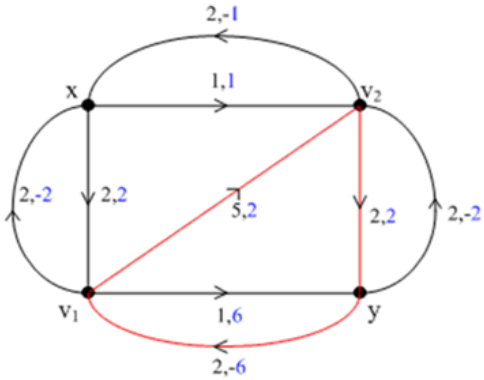
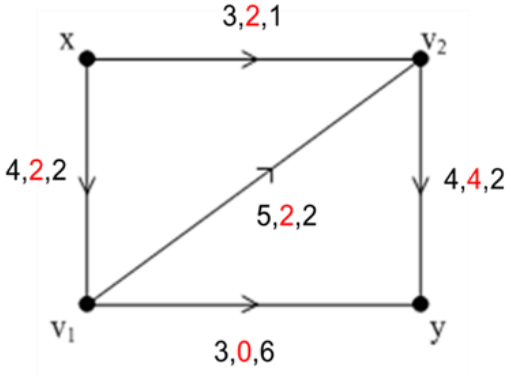
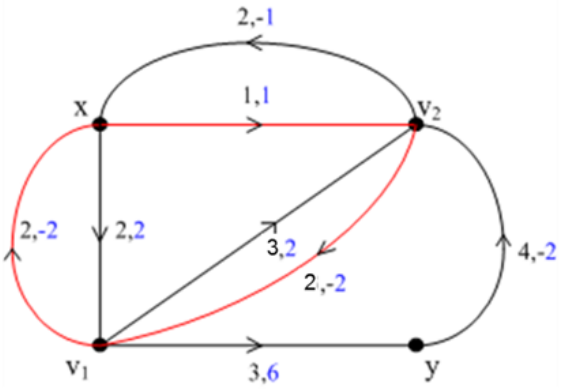
- 若不存在Q， 则f即为所求的最小费用流，算法终止。

在这个过程中有两个关键点：

① 按N_f中的回路容量调整f;

② f为N上流值λ的最小代价流的充要条件：在N_f中不存在负回路。

例题：已知网络N上的流f, 流值$Val\ f$=4，求最小费用流，弧上的数字$u(e)$、$f(e)$、$W(e)$分别表示边容量、流值和权重。

网络 N u(e)，f(e)，W(e)	增流网络 N_f u'(e)，W'(e)
	
$Val\,f$=4　　$W(f)$=28	调整量　δ=min{4,3,2}=2 负回路费用　-2-2+1 = -3
	
$Val\,f$=4 $W(f)$=28+2*(-3)=22	调整量　δ=min{5,2,2}=2 负回路费用　2+2-6 = -2
	
$Val\,f$=4 $W(f)$=22+2*(-2)=18	调整量　δ=min{2,2,1}=1 负回路费用　-2+1-2 = -3

算法二：给定最小费用流，提高流值到λ.

算法步骤如下：

①. 取初始流*f*为零流。

②. 判断 $Val\,f=\lambda$ ？

- 若是， 则f即为N中流值为λ的最小费用流，算法终止。
- 若否，则作增流网络N*f*.

③. N_f中是否存在一条从*x*到y的路径？

- 若是，则找一条最短路径P。调整量δ为：

$$\delta = min\{u'(P), \lambda - Valf\}$$

在N上得到调整后的流$\tilde{f}$

$$\tilde{f} = \begin{cases} f(e)+\delta & e\in P,\text{前向边} \\ f(e)-\delta & e\in P,\ \text{后向边} \\ f(e) & e\notin P \end{cases}$$

$$
\tilde{f} = \begin{cases} f(e)+\delta\quad e\in P,\text{前向边} \\ f(e)-\delta\quad e\in P,\text{后向边} \\ f(e)\quad e\notin P \end{cases}
$$

Error: You can't use 'macro parameter character #' in math mode

$$
Maximize\quad \sum_{i=1}^m\sum_{j=1}^mc_{ij}x_{ij}\\
s.t.\quad \sum_{j=1}^{m}x_{ij}-\sum_{k=1}^{m}x_{ki}=b_i=\begin{cases}q,\quad if\quad i=1\\0,\quad if\quad i\neq1\quad i\neq m\\-q, \quad if\quad i=m\end{cases}\\
0\leq x_{ij}\leq u_{ij}\quad i,j\in\{1,...,m\}
$$

第一个约束条件中的第一求和项表示由节点i流出的流量和；第二求和项表示流入节点i的流量之和。对比前面提高的最短路线性规划模型，其实最短路问题可视为最小费用流问题的特殊情况，即边容量均为1的情形。

其他应用

最小费用流问题除了在运输问题中有较多的应用外，还有许多别的应用场景，此处仅介绍建模思路，求解过程略。

- 缺货问题

  现有3个汽车配件厂x1、x2和x3，欲将配件运往3个汽车修配厂y1、y2、y3。若修配厂需要的配件得不到满足，就要形成缺货损失费。设y1处不能缺货， y2、y3处每缺一个单位配件就分别造成缺货损失费2和3。其它有关参数如下表。问如何调配，使总的费用最低？

W_{ij}	y_1	y_2	y_3	供应
x_1	2	1	3	5
x_2	2	-	4	3
x_3	-	4	3	5
需求	4	6	5	

  缺货问题实际上是运输问题的一种，是供给（13: 5+3+5）<需求（15: 4+6+5）的运输问题。我们通过引入虚拟节点和虚拟边，可以将这类缺货问题转换为供需平衡的运输问题。

  据题意，我们构建了图6.27所示的网络图模型（单源单汇网络）。边上的参数为（Cij, Wij）。因为缺货2个单位，故设置一个虚拟的供应点x4，因y1不能缺货，y2,y3允许缺货，故设置有向边（x4, y2），（x4, y3）的边容量为2，Wij为缺货费2和3。若求解结果显示f（x4, y2）> 0，这意味着y2处存在缺货f（x4, y2）单位，y2实际收量为6- f（x4, y2）。若f（x4, y2）= 0，这意味着y2处不缺货。

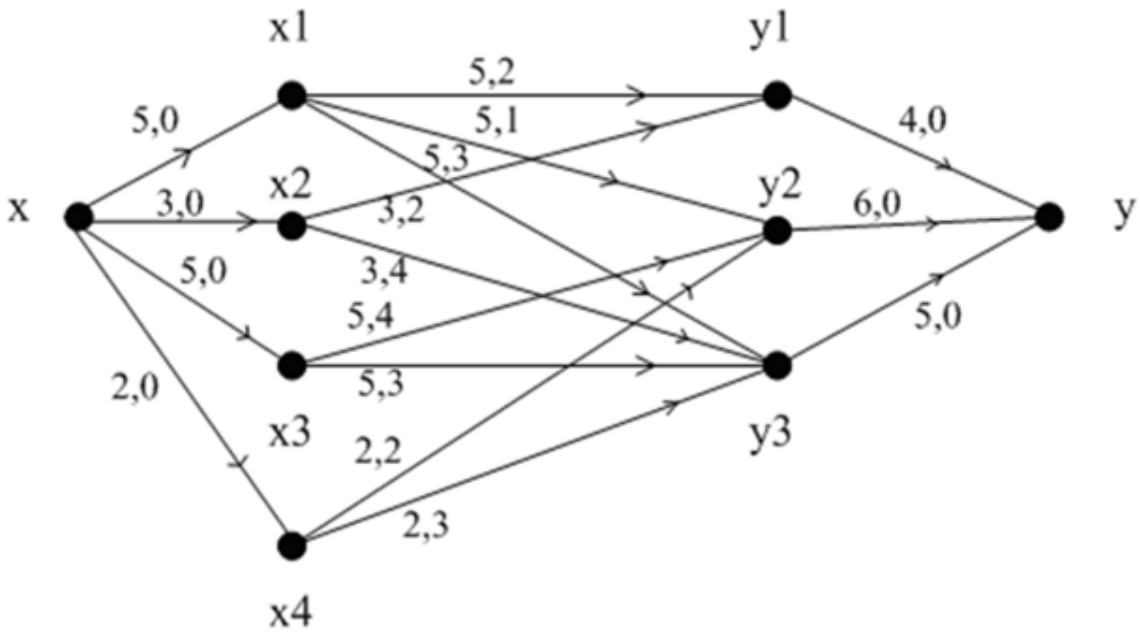


图6.27 缺货问题的网络图模型

- 生产计划问题

某工厂明年根据合同，每个季度末向销售公司提供产品，有关信息如下表。若当季生成的产品过多，季末有积余，则一个季度每积压一顿产品需支付存储费0.2万元。现该厂考虑明年的最佳生产方案，使该厂在完成合同的情况下，全年的生产费用最低。

季度j	生产能力a_j	生产成本d_j	需求量b_j
1	30	15.0	20
2	40	14.0	20
3	20	15.3	30
4	10	14.8	10

设x为源（工厂），y为汇（销售公司），vj为第j季度产品的存储与供货点（j=1,2,3,4）。我们将每季度的生产能力aj设为边（x, vj）的容量C_{xv_j},生产能力dj设为边（x, vj）的费用 , 需求量bj设为边（vj, y）的容量 ，无成本，故 =0。考虑部分产品可能积压一个季度，需付存储费0.2万元，故有 = 0.2. 得到的网络图模型如图6.28所示。

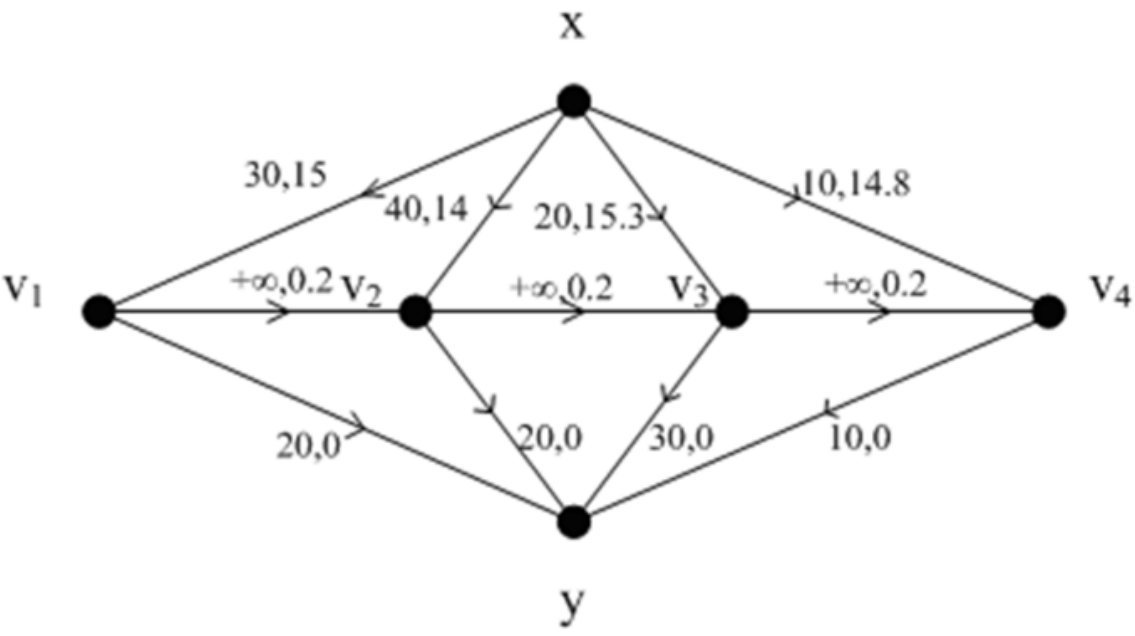


图6.28 生产计划问题的网络图模型

参考文献

[1] Bertsimas, Dimitris, and John N. Tsitsiklis. Introduction to linear optimization. Vol. 6. Belmont, MA: Athena Scientific, 1997.

[2] Bazaraa M S , Jarvis J J , Sherali H D . Linear programming and network flows[M]. WILEY, 1977.

[3] Ahuja R K, Magnanti T L, Orlin J B. Network flows: theory, algorithms, and applications[J]. Journal of the Operational Research Society, 1993, 45(11):791-796.

[4] Goldberg A V, Radzik T. A heuristic improvement of the Bellman-Ford algorithm[J]. Applied Mathematics Letters, 1993, 6(3):3-6.

[5] Bundy A, Wallen L. A* Algorithm [M] Catalogue of Artificial Intelligence Tools. 1984.

[6] Hillier F S. Introduction to operations research [M]. Tata McGraw-Hill Education, 2012.

[7] 马进.运筹学[M].北京:人民交通出版社，2003:186-192

[8] 傅家良.运筹学方法与模型 第2版[M].上海：复旦大学出版社.2014.

第7章 优化求解器及其比较

作者：魏晓阳 新加坡国立大学 土木与环境工程系 博士生在读

1. 优化求解器

1.1. AIMMS

AIMMS（Advanced Integrated Multidimensional Modeling System，
高级交互式多维建模系统）有两个主要产品，可提供跨多个行业的建模和优化功能：(1) AIMMS规范分析平台允许高级用户开发基于优化的应用程序并将其部署到业务用户；(2) AIMMS SC
Navigator基于AIMMS规范分析平台构建，为供应链团队提供可配置的应用程序，为非高级用户提供了供应链分析。AIMMS旨在建模和解决大规模优化和调度问题，被认为是五种最重要的代数建模语言之一，该软件获得了INFORMS影响力奖。

AIMMS规范分析平台包括代数建模语言，用于编辑模型和围绕这些模型创建图形用户界面的集成开发环境，以及图形最终用户环境。随着对用于供应链管理的嵌入式高级分析的兴趣日益浓厚，AIMSS开发了AIMMS
SC
Navigator平台以支持供应链分析，最初使用三个基于云的应用程序：供应链网络设计、销售和运营计划以及数据导航器。

AIMMS通过AIMMS开放求解器接口链接到多个求解器。支持的求解器包括CPLEX、MOSEK、FICO
Xpress、CBC、Conopt、MINOS、IPOPT、SNOPT、KNITRO和CP Optimizer。

AIMMS具有声明式和命令式programming风格的混合体。优化模型的制定是通过声明性语言元素（例如集合和索引，以及标量和多维参数，变量和约束）进行的，这些元素对于所有代数建模语言都是通用的，并且可以简要描述大多数问题的数学优化。语言本身就支持度量单位，并且可以使用编译和运行时单位分析来检测建模错误。为了支持通用建模组件的重用，AIMMS允许建模人员在用户模型库中组织其模型。AIMMS支持多种数学优化问题类型：

(1) 线性规划

(2) 二次规划

(3) 非线性规划

(4) 混合整数规划

(5) 混合整数非线性规划

(6) 全局优化互补问题（MPEC）

(7) 随机规划

(8) 鲁棒优化

(9) 约束规划

通过指定其他属性，可以在AIMMS中的确定性线性和混合整数优化模型中考虑不确定性，从而可以将随机性或鲁棒性优化技术与现有的确定性解决方案技术一起使用。可以使用GMP系统库构建自定义的混合算法和分解算法，该库在建模级别提供AIMMS中存在的高级解决方案方法，矩阵修改方法以及用于自定义解决方案的特定问题类型的算法。用AIMMS创建的优化解决方案既可以用作独立的桌面应用程序，也可以作为软件组件嵌入其他应用程序中。

1.2. ALGLIB

ALGLIB是一个跨平台的开源数值分析和数据处理库。可以通过多种编程语言（C ++、C＃、VB.NET、Python、Delphi）调用它。它被多个开源项目、商业库和应用程序使用（例如TOL项目、Math.NET Numerics、SpaceClaim）。该库的独特功能是：

(1) 支持具有相同API的多种编程语言（C ++、C＃、FreePascal / Delphi、VB.NET和Python）

(2) 自包含代码，没有强制性的外部依赖关系，易于安装

(3) 可移植性（已在x86 / x86-64 / ARM、Windows和Linux下进行了测试）

(4) 两个独立的后端（纯C＃实现、本机C实现）和自动生成的API（C ++、C＃等）

(5) 商业版和GPL版具有相同的功能

ALGLIB提供以下功能：

(1) 线性代数（直接算法、求解器、EVD / SVD）

(2) 快速傅里叶变换

(3) 数值积分

(4) 插值

(5) 线性和非线性最小二乘拟合

(6) 优化

(7) 常微分方程

(8) 统计（描述性统计、假设检验）

(9) 数据分析（分类/回归，包括神经网络）

(10) 线性代数、插值和优化算法的多种精度版本（使用MPFR进行浮点计算）

1.3. AMPL

数学编程语言（AMPL）是一种代数建模语言，用于描述和解决大规模数学计算的高复杂性问题（即大规模优化和调度问题）。它是由Bell实验室的Robert
Fourer、David Gay和Brian Kernighan开发的。
AMPL支持许多求解器，包括CBC、CPLEX、FortMP、Gurobi、MINOS、IPOPT、SNOPT、KNITRO和LGO。问题以nl文件的形式传递给求解器。

AMPL的优点之一是其语法与优化问题的数学符号相似。这允许在优化领域中对问题进行非常简明易懂的定义。
NEOS服务器（以前由Argonne国家实验室托管，目前由威斯康星大学麦迪逊分校托管）上可用的许多现代求解器都接受AMPL输入。根据NEOS统计，AMPL是表示数学编程问题的最流行格式。AMPL混合了声明式和命令式编程样式。建立优化模型是通过声明性语言元素（例如集合、标量、多维参数、决策变量、目标和约束条件）进行的，从而可以对数学优化领域中的大多数问题进行简洁的描述。为了支持重用并简化大规模优化问题的构建，AMPL允许模型和数据分离。
AMPL支持多种问题类型，其中包括：

(1) 线性规划

(2) 二次规划

(3) 非线性规划

(4) 混合整数规划

(5) 具有或不具有凸二次约束的混合整数二次规划

(6) 混合整数

(7) 非线性规划

(8) 二阶锥规划全局优化

(9) 具有双线性矩阵不等式的半定规划问题

(10) 离散或连续变量的互补理论问题（MPEC）

(11) 约束规划

1.4. ANTIGONE

ANTIGONE（Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations）是混合整数非线性程序（MINLP）的确定性全局优化求解器。ANTIGONE是GlomiQO的演进，它是Ruth
Misener编写的全局混合整数二次规划求解器。
ANTIGONE将GloMIQO的功能扩展到了一般的MINLP问题。

像所有确定性全局优化软件一样，ANTIGONE是许多技术的工具箱，用于处理非线性结构的不同特殊情况。话虽如此，它主要是分支定界求解器。它的主要算法过程分为4个主要步骤：

(1) 用户输入的重新形成

(2) 特殊结构的检测

(3) 为检测到的结构选择最佳算法

(4) 使用所选算法解决问题

除了特殊情况下的优化问题（例如凸NLP）可以在分支定界算法的根节点处求解，
ANTIGONE将启动分支定界过程。此过程包括以下步骤：

(1) 生成/更新凸松弛（下界问题）

(2) 域约简

(3) 搜索可行的解决方案（上限）

(4) 通过解决凸下界问题来计算严格的下界

ANTIGONE采用经典的可分解programming技术来生成relaxation（例如McCormick relaxation），以及重新线性化技术（RLT）、边缘凸/凹relaxation和Alpha BB切割。
ANTIGONE还拥有一个动态剪切生成器，它可以生成并处理本地和全局有效剪切。

像所有确定性全局优化软件一样，ANTIGONE要求用户为问题中使用的所有函数提供明确的数学表达式，以及所有变量的初始边界。如果未提供初始界限，则ANTIGONE将尝试推断界限，但不能保证全局最优性。
ANTIGONE只能解决微分函数，不能解决三角问题。ANTIGONE是GAMS建模平台的一部分。

1.5. APMonitor

高级过程监控器（APMonitor）是一种用于微分代数（DAE）方程的建模语言。它是一个免费的Web服务或本地服务器，用于以隐式DAE模型的形式求解物理系统的表示形式。
APMonitor适用于大规模问题，可解决线性规划、整数规划、非线性规划、非线性混合整数规划、动态仿真、动视域估计和非线性模型预测控制。APMonitor不能直接解决问题，而是调用非线性规划求解器，例如APOPT、BPOPT、IPOPT、MINOS和SNOPT。
APMonitor
API通过自动微分和稀疏矩阵形式为求解器提供连续函数的精确一阶和二阶导数。

Julia、MATLAB和Python等编程语言，通过Web服务API集成了APMonitor。
GEKKO优化套件是APMonitor的最新扩展，具有完整的Python集成。这些接口是内置的优化工具箱或模块，用于加载和处理优化问题的解决方案。
APMonitor是一种面向对象的建模语言和优化套件，它依赖于编程语言来加载、运行和检索解决方案。
APMonitor模型和数据在运行时进行编译，并转换为对象，这些对象由优化引擎（例如APOPT或IPOPT）解决。
APMonitor未指定优化引擎，因此可以切换多种不同的优化引擎。还可以配置仿真或优化模式，以重新配置模型、进行动态仿真、非线性模型预测控制、移动视界估计或数学优化中的一般问题。

1.6. APOPT

APOPT（Advanced Process
OPTimizer）是用于解决以下任意形式的大规模优化问题的软件包：

(1) 线性规划（LP）

(2) 二次规划（QP）

(3) 二次约束二次规划（QCQP）

(4) 非线性规划（NLP）

(5) 混合整数规划（ MIP）

(6) 混合整数线性规划（MILP）

(7) 混合整数非线性规划（MINLP）

APOPT的应用包括化学反应器、摩擦搅拌焊接、防止深海管道中水合物的形成、计算生物学、固体氧化物燃料电池和无人飞行器（UAV）的飞行控制。

1.7. Artelys Knitro

Artelys Knitro 是用于解决大规模非线性数学优化问题的商业软件包。 KNITRO由Richard Waltz、Jorge Nocedal、Todd Plantenga和Richard
Byrd共同创建。它于2001年作为美国西北大学学术研究的衍生产品（通过Ziena Optimization LLC）首次引入，此后一直由Artelys的开发人员不断改进。

优化问题必须以数学形式呈现给Knitro，并应提供一种使用稀疏矩阵计算函数导数的方式（Knitro可以计算导数近似值，但在大多数情况下，提供精确的导数是对求解有帮助的）。通常更容易的方法是用代数建模语言搭建优化问题。Knitro专门从事非线性优化，但也解决了一系列优化问题：

(1) 一般非线性问题（NLP），包括非凸问题

(2) 非线性方程组

(3) 线性问题（LP）

(4) 凸和非凸二次问题（QP / QCQP / SOCP）

(5) 最小二乘问题/线性和非线性回归

(6) 具有互补性约束的数学程序（MPCC / MPEC）

(7) 混合整数非线性问题（MIP / MINLP）

(8) 无导数优化问题（DFO）

Artelys Knitro包含多种优化算法：

(1)
非线性规划（NLP）求解器。Knitro提供了四种不同的优化算法来解决优化问题。两种算法属于内部点类型，两种算法属于活动集（active
set）类型。众所周知，这些算法具有根本不同的特征。例如，内部点方法沿着通过可行区域内部的路径行进，而活动集（active
set）方法往往停留在边界处。
Knitro可以在求解过程中从一种算法转换为另一种算法。该代码还提供了一个多启动选项，以促进全局最小值的计算。

(i) 内部/直接算法

(ii) 内部/共轭梯度算法

(iii) 活动集（active set）算法

(iv) 顺序二次规划（SQP）算法

(2)
混合整数非线性规划（MINLP）求解器。Knitro提供了用于求解具有二进制或整数变量的优化模型（线性和非线性）的工具。
Knitro混合整数规划（MIP）代码为混合整数非线性规划（MINLP）提供了三种算法：

(i) 非线性分支和边界

(ii) Quesada Grossman算法

(iii) 混合整数顺序二次规划（MISQP）

Artelys Knitro支持多种编程和建模语言，包括：

(1) 适用于C ++、C＃、Java和Python的面向对象的接口

(2) 适用于C、Fortran、MATLAB和R的面向矩阵的接口

(3) 链接到建模语言AIMMS、AMPL、GAMS和MPL

(4) 通过Frontline Solvers链接到Excel

Artelys Knitro还包括许多关键功能：

(1) 大量用户选项和自动调谐器

(2) （并行）全局优化的多起点

(3) 导数近似和校验器

(4) 内部预解析器

1.8. Cassowary

Cassowary是一个增量约束解决工具包，可以有效地解决线性等式和不等式的系统。约束可以是要求，也可以是首选项。客户代码指定要维护的约束，求解器将约束变量更新为具有满足约束的值。

Cassowary由Greg Badros、Alan Borning和Peter J.
Stuckey开发，并针对用户界面应用进行了优化。Badros还使用Cassowary来实现约束层叠样式表（CCSS），这是层叠样式表（CSS）的扩展。CCSS添加了对布局约束的支持。这些使设计人员可以更灵活地描述网页的布局。Cassowary用于解决这些限制并计算最终布局。

主版本中提供Smalltalk、C ++和Java版本。此外，还有GNU
Guile、Python和STk的绑定。其他人已将求解器移植到JavaScript、Dart、Squeak、Python、.NET框架（Cassowary.net）和Rust。

1.9. COIN-OR

Computational Infrastructure for Operations Research（COIN-OR）是一个旨在“create for mathematical software what the open literature is for mathematical theory”。开放文献（例如研究期刊）为运筹学（OR）社区提供了同行评审过程和存档。运筹学期刊上有关数学理论的论文通常包含支持计算研究的数值结果。通常不会发布用于产生数值结果的软件实现，模型和数据。这一现状阻碍了研究人员重现计算结果，进行公平的比较并扩展了现有技术水平。
COIN-OR被认为是一项计划，目的是在计算运筹研究社区中推广开源软件，并提供使其他人能够运行自己的开源软件项目所需的在线资源和托管服务。

COIN-OR网站是与2000年在佐治亚州亚特兰大举行的第17届国际数学编程专题讨论会一起启动。在2007年，COIN-OR拥有25个应用项目，包括用于线性规划（例如COIN-OR CLP）、非线性规划（例如IPOPT）、整数规划（例如CBC、Bcp和COIN-OR SYMPHONY）的工具、代数建模语言（例如Coopr）等。到2011年，该项目已发展到48个

项目。COIN-OR由INFORMS运筹学与管理科学研究所主办，由教育性、非营利性COIN-OR基金会运营。

在AIMMS、AMPL和GAMS建模系统以及FortSP求解器中都可以使用COIN-OR求解器。也可以通过OpenSolver和SolverStudio加载项在Excel中使用它们。

1.10. CPLEX

IBM ILOG CPLEX Optimization Studio（CPLEX）是一个优化软件包。
2004年，CPLEX的工作获得了首届INFORMS影响力奖。CPLEX
Optimizer以使用C编程语言实现的单纯形方法而命名，尽管今天它还支持其他类型的数学优化并提供除C之外的接口。它最初由Robert
E.
Bixby开发，并于1988年由CPLEX商业出售，随后ILOG在2009年1月被IBM收购。IBM继续积极开发CPLEX。

IBM ILOG CPLEX
Optimizer解决了整数规划问题、非常大的线性规划问题、凸和非凸二次规划问题以及凸二次约束问题（通过二阶锥规划或SOCP求解）。
CPLEX Optimizer具有一个称为Concert的建模层，该层提供了与C
++、C＃和Java语言的接口，有一个基于C接口的Python语言接口。此外，还提供了Microsoft
Excel和MATLAB的连接器。最后，提供了一个独立的Interactive
Optimizer可执行文件，用于调试和其他目的。

可通过独立的建模系统（例如AIMMS、AMPL、GAMS、OptimJ和TOMLAB）访问CPLEX Optimizer。除此以外，AMPL还提供了CPLEX CP Optimizer的接口。完整的IBM ILOG CPLEX
Optimization Studio包括用于数学规划的CPLEX Optimizer、用于约束规划的CP Optimizer、优化编程语言（OPL）和集成的IDE。

1.11. FICO Xpress

FICO
Xpress优化器用于线性规划（LP）、混合整数线性规划（MILP）、凸二次规划（QP）、凸二次约束二次规划（QCQP）、二阶锥规划（SOCP）等。Xpress包括通用非线性求解器Xpress
NonLinear，其中包括连续线性规划算法（一阶方法）和Artelys Knitro（二阶方法）。

Xpress最初由Dash
Optimization开发，并于2008年被FICO收购。它的最初作者是鲍勃·丹尼尔（Bob
Daniel）和罗伯特·阿什福德（Robert Ashford）。
Xpress是第一个通过在2010年引入64位索引来突破十亿决策变量阈值的MIP解决方案。自2014年以来，Xpress首次采用并行对偶单纯形法的商业实现。

线性和二次程序可以通过原始单纯形法、对偶单纯形法或barrier interior point
求解。所有混合整数规划变量都通过分支定界方法和切面方法的组合来求解。可以通过IIS（Irreducible
Infeasible Subset）方法分析不可行的问题。
Xpress提供了一个内置的调谐器，用于自动调整控制设置。Xpress包括其建模语言Xpress Mosel和集成开发环境Xpress Workbench。
Mosel包括分布式计算功能，因此可以并行解决优化问题的多种情况。

Xpress具有一个称为BCL（Builder Component Library）的建模模块，该模块可与C、C++、Java编程语言和.NET框架接口。独立于BCL，有Python和MATLAB接口。
Xpress还可以连接到其他标准建模语言，例如AIMMS、AMPL和GAMS。FICO Xpress Executor使用SOAP或REST接口执行和部署Mosel模型。可以从外部应用程序或FICO Decision
Management Platform使用它。

1.12. FortMP

FortMP是用于解决大规模优化问题的软件包。它解决了线性规划问题、二次规划问题和混合整数规划问题（线性和二次）。它的鲁棒性已经被探索并发表在《Mathematical Programming》上。

FortMP可以作为独立的可执行文件接受MPS格式的输入，也可以作为具有C和Fortran接口的库提供。
AMPL建模系统也支持它。FortMP中实现的主要算法是使用稀疏矩阵的原始和对偶单纯形算法。通过内点法对大型问题和二次规划问题进行了补充。使用分支定界算法解决混合整数规划问题。

1.13. Gekko

GEKKO
Python软件包使用非线性规划求解器（IPOPT、APOPT、BPOPT、SNOPT和MINOS）求解大型混合整数和微分代数方程。操作模式包括机器学习、数据协调、实时优化、动态仿真和非线性模型预测控制。此外，该软件包还解决了线性规划（LP）、二次规划（QP）、二次约束二次规划（QCQP）、非线性规划（NLP）、混合整数规划（MIP）和混合整数线性规划（MILP）的问题。

GEKKO可以被Python中调用。默认情况下，问题将发送到公共服务器，在该服务器上计算解决方案并将其返回给Python。可以在Windows、MacOS、Linux和ARM（Raspberry Pi）平台上运行。 GEKKO是APMonitor Optimization Suite的扩展，但已将建模和解决方案可视化，直接集成在Python中。

应用包括热电联产（电力和热力）、钻探自动化、严格的炉渣控制、太阳能项目、固体氧化物燃料电池、流量保证、提高采油率、提取精油和无人飞行器（UAV）。GEKKO是由美国国家科学基金会（NSF）的研究资助开发的，在有关联合调度与控制的特刊中有详细介绍。

1.14. GLPK/GLPSOL

GNU Linear Programming Kit（GLPK）是一个软件包，用于解决大规模线性规划（LP）、混合整数规划（MIP）和其他相关问题。它是一组用ANSI C编写并以可调用库的形式组织的例程。该软件包是GNU项目的一部分，并根据GNU通用公共许可证发行。

可以使用GNU MathProg语言（以前称为GMPL）对问题进行建模，该语言与AMPL共享大多数语法，并使用独立的求解器GLPSOL进行解决。GLPK也可以用作C库。

GLPK对非整数问题使用修订的单纯形法和原始对偶内点法，并将分支定界算法与Gomory的混合整数割对（混合）整数问题结合使用。免费版的OptimJ建模系统支持GLPK，这允许Java应用程序以相对透明的方式调出GLPK。GLPK由莫斯科航空学院的Andrew O. Makhorin开发，首次公开发布于2000年10月。

1.15. GNU Scientific Library

GNU Scientific Library（GSL）是一个用于在应用数学和科学中进行数值计算的软件库。GSL用C编写。 GSL是GNU Project的一部分，并根据GNU通用公共许可证进行分发。

GSL项目由洛斯阿拉莫斯国家实验室的物理学家Mark Galassi和James Theiler于1996年发起。他们的目的是为广泛使用但有些过时的Fortran库（例如Netlib）编写一个现代的替代品。他们进行了总体设计并编写了早期模块。

由于GSL是用C编写的，因此很容易为其他编程语言提供wrapper。这些wrapper目前存在于：

(1) AMP

(2) C++

(3) Fortran

(4) Haskell

(5) Java

(6) Lisp

(7) Ocaml

(8) Octave

(9) Perl Data Language

(10) Python

(11) R

(12) Ruby

1.16. Gurobi

Gurobi
Optimizer是用于线性规划（LP）、二次规划（QP）、二次约束规划（QCP）、混合整数线性规划（MILP）、混合整数二次规划（MIQP）和混合整数二次规划（MIQCP）的商业优化求解器。

Gurobi成立于2008年，以其创始人命名：Zonghao **Gu**、Edward **Ro**thberg和Robert **Bi**xby。
Bixby还是CPLEX的创始人，而Rothberg和Gu领导CPLEX开发团队近十年。Gurobi
Optimizer支持多种编程和建模语言，包括：

(1) C ++、Java、.NET和Python的面向对象接口

(2) 适用于C、MATLAB和R的面向矩阵接口

(3) 可以链接到标准建模语言AIMMS、AMPL、GAMS和MPL

(4) 通过其分析求解器和求解器SDK产品链接到Excel

Gurobi Optimizer还包括许多功能来支持优化模型的构建，其中包括：

(1) 可以灵活确定多个目标的优先级

(2) 诸如MIN / MAX、ABS、AND / OR之类的常规约束以及指标约束

(3) 具有凸、分段线性目标函数的模型用于描述某些非线性问题

(4) 任意分段线性目标函数

(5) 分布式调参，以加快参数设置的探索速度，加快求解时间

Gurobi Optimizer还可以部署在云上，以及用于服务器-客户端模式。

1.17. IMSL Numerical Libraries

IMSL Numerical
Libraries是商业上具有数字分析功能的软件库，这些软件库以计算机编程语言C、Java、C＃.NET和Fortran实现，也提供Python界面。IMSL库由Rogue
Wave Software提供。

第一个针对Fortran语言的IMSL库于1970年发布，随后是1991年最初称为C /
Base的C语言版本，2002年的Java语言版本和2004年的C＃语言版本。最近的一些发行版都涉及Python提供IMSL库函数。PyIMSLwrapper程序于2008年8月首次发布。PyIMSL
Studio在2009年2月推出。PyIMSLStudio可免费下载用于非商业用途或用于商业评估。各种操作系统\硬件和编译器均支持IMSL
Numerical Libraries。

1.18. LINDO

LINDO（Linear, Interactive, and Discrete
Optimizer）是用于线性规划、整数规划、非线性规划、随机规划和全局优化的软件包。

Lindo还创建了“ What'sBest！”，是一种线性、整数和非线性优化的插件。最初针对Lotus 1-2-3发布，后来也针对Microsoft Excel发布。

1.19. LIONsolver

LIONsolver（Learning and Intelligent
OptimizatioN）是用于数据挖掘、商业智能、分析和建模的集成软件。非营利版本为LIONoso。LIONsolver可用于构建模型、可视化模型并改善业务和工程流程。它是基于数据和定量模型进行决策的工具，可以连接到大多数数据库和外部程序，并且与Grapheur商业智能软件完全集成，并且适合对设计业务逻辑和流程感兴趣的高级用户使用。

LIONsolver源自反应式搜索优化中的研究原理，该论点主张在软件系统运行时使用自调整方案。学习和智能优化是指将在线机器学习方案集成到优化软件中，从而使其能够从以前的运行和人工反馈中学习。它提供了一种定义设计空间的直接方法，其中涉及反应式搜索优化，而“自主搜索”则提倡采用自适应问题解决算法。建模组件包括神经网络、多项式、局部加权贝叶斯回归、k均值聚类和自组织图。提供了用于非商业用途和课堂使用的免费学术许可证。

LIONsolver的软件体系结构允许交互式和多目标优化，其用户界面可直观显示结果并促进解决方案分析和决策过程。该体系结构允许针对特定问题进行扩展，并且它可作为具有多种不同潜在解决方案的所有优化方案的后处理工具。当架构与特定的问题解决或优化方法紧密耦合时，可以开发有效的交互方案。

1.20. Math Kernel Library

Intel Math Kernel Library (Intel MKL)
是针对科学、工程和金融应用程序优化的数学例程的库。核心数学功能包括BLAS、LAPACK、ScaLAPACK、稀疏求解器、快速傅立叶变换和矢量数学。MKL中的例程专门针对Intel处理器进行了优化。该库支持Intel处理器，可用于Windows、Linux和macOS操作系统。

英特尔C
++编译器生成的英特尔MKL和其他程序使用一种称为函数多版本化的技术来提高性能。英特尔MKL具有以下功能类别：

(1) 线性代数

(2) 快速傅立叶变换（FFT）

(3) 向量数学

(4) 统计

(5) 数据拟合

(6) 深度神经网络

(7) 偏微分方程

(8) 非线性优化问题求解器

1.21. MathCad

MathCad是一种交互式数值计算系统。当输入一个数学公式、方程组、矩阵等，计算机将直接给出计算结果，而无须去考虑中间计算过程。因而MathCad在航空、国防、消费品设计等科学和工程领域中承担着复杂的数学计算、图形显示和文档处理，是工程技术人员常用的工具。Mathcad有五个扩展库，分别是求解与优化、数据分析、信号处理、图像处理和小波分析。

Mathcad采用接近在黑板上写公式的方式让用户表述所要求解的问题，通过底层引擎计算返回结果并显示在屏幕上。计算过程近似透明，使用户专注于对问题的思考而不是繁琐的求解步骤。

经过20年发展，Mathcad从早期的简单有限功能发展到现在的代数运算、线性及非线性方程求解与优化、常微分方程、偏微分方程、统计、金融、信号处理、图像处理等许多方面。并提供丰富的接口可以调用第三方软件的功能，利于自行扩展和利用别的软件扩展功能。Mathcad集programming、计算、显示、文档记录于一体，是美国PTC公司的产品。

1.22. Mathematica

Wolfram
Mathematica（通常称为Mathematica）是一个现代技术计算系统，涵盖了技术计算的大多数领域，包括神经网络、机器学习、图像处理、几何、数据科学、可视化等。该系统用于许多技术、科学、工程、数学和计算领域。它由斯蒂芬·沃尔夫拉姆（Stephen Wolfram）构思，并由伊利诺伊州尚佩恩的沃尔夫拉姆研究中心（Wolfram Research）开发。Wolfram语言是Mathematica中使用的编程语言。

Wolfram
Mathematica分为内核和前端两部分。内核解释表达式（Wolfram语言代码）并返回结果表达式。前端由Theodore Gray在1988年设计，提供了一个GUI，它允许创建和编辑Notebook文档，其中包含带有语法突出显示的程序代码、格式化的文本、结果、表格和声音。所有内容和格式都可以通过算法生成或进行交互式编辑。支持标准的文字处理功能，包括实时的多语言拼写检查。

可以使用单元的层次结构来结构化文档，从而可以对文档进行概述和分段，并支持自动编号索引的创建。笔记本及其内容以Mathematica表达式表示，可以由Mathematica程序创建、修改或分析，也可以转换为其他格式。演示者工具支持创建幻灯片样式的演示文稿，该演示文稿支持演示过程中的交互元素和代码执行。

Wolfram Workbench是可供选择的前端之一，Wolfram Workbench是基于Eclipse的集成开发环境（IDE），于2006年推出。它为Mathematica提供基于项目的代码开发工具，包括修订管理、调试、配置文件和测试。有一个基于IntelliJ IDEA的IDE可以与Wolfram语言代码一起使用的插件，该插件除了语法高亮显示之外还可以分析和自动完成局部变量和定义的函数。Mathematica内核还包括命令行前端，其他接口包

括基于GNU
readline的JMath 和从UNIX命令行运行的WolframScript。

以下方法可以部署用Wolfram Mathematica编写的应用程序：

(1) Mathematica Player
Pro是Mathematica的运行时版本，它将运行任何Mathematica应用程序，但不允许编辑或创建代码。

(2) 提供了免费版本Wolfram CDF
Player，用于运行以可计算文档格式（CDF）保存的Mathematica程序。它也可以查看标准Mathematica文件，但不能运行它们。它包括适用于Windows和Macintosh上常见Web浏览器的插件。

(3)
webMathematica允许Web浏览器充当远程Mathematica服务器的前端。它旨在允许通过任何平台上的浏览器远程访问用户编写的应用程序。它可能无法用于完全访问Mathematica。由于带宽限制，Web浏览器不完全支持交互式3D图形。

(4) Wolfram语言代码可以转换为C代码或自动生成的DLL。

(5) Wolfram语言代码Web应用程序或API在Wolfram托管的服务器上或Wolfram Enterprise Private Cloud的私有安装中运行。

1.23. MIDACO

MIDACO（Mixed Integer Distributed Ant Colony
Optimization）是用于基于进化计算的数值优化的软件包。 MIDACO是与欧洲航天局和EADS
Astrium合作创建的，旨在解决受限的混合整数非线性（MINLP）空间应用。MIDACO拥有欧洲航天局公开提供的关于行星际航天轨迹设计问题的几种记录解决方案。
MIDACO包含在以下软件包中：TOMLAB、Astos和SigmaXL等。

1.24. MINOS

MINOS是用于解决线性和非线性数学优化问题的Fortran软件包。
MINOS（可用于线性规划、二次规划以及更通用的目标函数和约束条件，以及为一组线性或非线性等式和不等式寻找可行点。

MINOS由Bruce Murtagh和Michael
Saunders首先开发，主要是由斯坦福大学运筹学系的系统优化实验室开发。1985年，桑德斯（Saunders）因在MINOS上的工作而被数学规划协会（现为数学优化协会）授予了首届Orchard-Hays奖。它是最先出现的通用约束优化求解器之一，现在仍在大量使用。
MINOS在AIMMS、AMPL、APMonitor、GAMS和TOMLAB建模系统中受支持。此外，它仍然是NEOS
Server和GAMS中使用次数最多的求解器之一。

理想情况下，用户应提供非线性函数的梯度。如果未提供某些或全部梯度，则MINOS将通过有限的差值来近似缺少的梯度，但这可能会很慢且可靠性较低。如果目标函数是凸的且约束是线性的，则获得的解将是全局极小值。否则，获得的解决方案可能是局部最小值。

对于线性程序，使用两阶段原始单纯形法。第一阶段将不可能性的总和最小化。对于具有线性约束和非线性目标的问题，使用减小梯度法。保持对简化的Hessian的拟牛顿近似，以获得搜索方向。当在解决方案上有许多约束或界限处于活动状态时，该方法最有效。对于具有非线性约束的问题，使用了线性约束拉格朗日方法。这涉及一系列主要迭代，每个迭代都（近似）解决一个线性约束子问题。子问题目标是扩充的拉格朗日，而子问题约束是当前点处非线性约束的线性化。MINOS旨在解决大型稀疏问题，问题大小没有固定的限制。源代码适用于所有具有Fortran编译器的科学机器。

1.25. MINTO

MINTO（Mixed Integer
Optimizer）是使用分支定界算法的整数规划求解器。MINTO是一个软件系统，它通过具有线性规划松弛的分支定界算法来解决混合整数规划问题。它还提供自动约束分类、预处理、原始启发式和约束生成。它还具有内置的剪切生成，可以创建背包剪切、GUB剪切、集团剪切、隐含剪切、流剪切、混合整数舍入和Gomory剪切。此外，用户可以通过提供各种可以定制MINTO的专用例程来丰富基本算法，以提高问题求解的效率。

MINTO本身没有线性规划（LP）求解器。它可以通过COIN-OR的OSI接口使用大多数LP求解器，例如CLP、CPLEX、XPRESS。它可以在Linux和Windows操作系统上运行，
MINTO是一种非商业解决方案，其可执行文件可以从其主页COR\@ L免费下载。

1.26. MOSEK

MOSEK是用于解决线性、混合整数线性、二次、混合整数二次、二次约束、圆锥和凸非线性数学优化问题的软件包。
MOSEK的重点是解决大规模稀疏问题，尤其是线性，圆锥二次（又称二阶锥规划）和半定（即半定规划）的内点优化器。该软件特别有效地解决了后一类问题。MOSEK内部点优化器的一个特殊功能是它基于所谓的均质模型。这意味着MOSEK可以可靠地检测到原始和/或双重不可行状态。

该软件由丹麦公司Mosek ApS开发，该公司由Erling D.
Andersen于1997年成立。除了内部点优化器之外，MOSEK还包括：

(1) 线性问题的原始和对偶单纯形优化器

(2) 线性、二次和圆锥问题的混合整数优化器

在版本9中，Mosek在其求解器中引入了对指数锥和幂锥的支持。该软件还提供与C、C＃、Java和Python语言的接口。大多数主要的建模系统都与MOSEK兼容，例如AMPL和GAMS。MOSEK也可以从诸如MATLAB和R编程语言/软件环境之类的流行工具中使用。

1.27. NAG Numerical Library

NAG Numerical
Library是由数值算法组开发和销售的软件产品。它是一个数值分析例程的软件库，其中包含1900多种数学和统计算法。该库涵盖的领域包括线性代数、优化、正交、常微分方程、偏微分方程、回归分析和时间序列分析。

NAG库的用户可以从其应用程序中调用其例程，以合并其数学或统计功能并解决数值问题，例如，找到函数的最小值或最大值、将曲线或曲面拟合到数据、求解微分方程。该库有多种形式，分别是NAG
C库、NAG
Fortran库和用于.NET的NAG库。可从几种计算环境访问其内容，包括标准语言（例如C、C++、Fortran、Visual
Basic、Java、Python和C＃）以及软件包（例如MATLAB、R、LabVIEW、Excel、Origin和Ch）。支持的操作系统包括Windows、Linux和macOS，以及Solaris、AIX和HP-UX。

1.28. NMath

NMath是Microsoft .NET Framework的数值程序包。它由CenterSpace
Software开发，基于MKL（英特尔的数字库）构建。

NMath包含矢量和矩阵类、复数、分解、线性规划、最小化、求根、结构化、稀疏矩阵、最小二乘、多项式、模拟退火、曲线拟合、数值积分和微分等。

1.29. Octeract Engine

Octeract
Engine是专有的大规模、并行、确定性、全局优化求解器，适用于一般的混合整数非线性程序（MINLP）。它使用MPI作为加速求解时间的一种方式。Octeract
Engine的第一个公开测试版于2019年8月发布。截至2019年12月，该引擎仍处于beta版。

Octeract
Engine是一个符号分支定界求解器。它是目前唯一支持超级计算的确定性全局优化软件。它的一些功能是：

(1) 通过MPI进行分布式计算

(2) 支持不连续的基本功能（例如最小和最大）

(3) 支持三角函数

(4) 保证全局最优

(5) 重新编写用户输入

(6) 检测特殊结构

(7) 通过区间算术和任意精度算术保证计算

(8) 基于Python的界面称为Octeract Shell

像所有确定性全局优化软件一样，Octeract
Engine需要针对问题中使用的所有函数使用明确的数学表达式。Octeract
Engine可以直接运行，也可以在C
++和Python中作为库调用。它支持以下建模语言：AMPL、ASL、GAMS、Pyomo。该引擎还具有用于以下求解器的接口：Gurobi、CBC、CLP、IPOPT、NLOPT、BONMIN。

1.30. OptaPlanner

OptaPlanner是用Java编写的开放源代码约束求解器。它通过构造启发式算法和元启发式算法解决了约束满足问题。OptaPlanner由Red
Hat赞助，后者是JBoss社区的一部分，并且与KIE集团的Drools和jBPM项目密切相关。

它由Geoffrey De
Smet于2006年以Taseree的名义创立。2013年3月，更名为OptaPlanner。它由一个专门的核心团队（由Red
Hat雇用）和外部社区贡献者进行持续开发。Red Hat针对OptaPlanner的支持产品称为Red Hat业务优化器（在2018年之前称为Red Hat
JBoss业务资源规划器）。OptaPlanner的参与者经常在研究竞赛中与学术研究人员竞争，获得ICON算法选择挑战赛（2014年）第二名。

1.31. Pyomo

Pyomo是用于制定优化模型的Python软件包的集合。Pyomo由Sandia国家实验室的William Hart、Jean-Paul Watson和加州大学戴维斯分校的David Woodruff共同开发。
Pyomo的重要扩展由Sandia国家实验室的Bethany Nicholson和John Siirola、普渡大学的Carl Laird和Gabriel Hackebeil开发。
Pyomo是一个开放源代码项目，可免费获得，并且已获得BSD许可。
Pyomo是COIN-OR项目的一部分，
Pyomo是一种流行的开源软件包，许多政府机构和学术机构都在使用。

Pyomo允许用户以类似数学优化中常用符号的方式在Python中制定优化问题。
Pyomo支持制定优化模型的面向对象样式，该模型由各种建模组件定义：集合、标量和多维参数、决策变量、目标、约束、方程式、析取等。可以使用python数据初始化优化模型，并且可以使用电子表格、数据库和各种格式的文本文件定义外部数据源。
Pyomo支持既没有数据定义的抽象模型，又有数据定义的具体模型。在这两种情况下，Pyomo都可以分离模型和数据。

Pyomo支持数十种开源和商业求解器，包括由AMPL、PICO、CBC、CPLEX、IPOPT、Gurobi和GLPK支持的许多求解器。
Pyomo可以直接调用求解器，也可以与求解器管理器异步调用。求解器管理器支持求解器的远程异步执行，该求解器支持Pyomo脚本的并行执行。求解器交互是通过各种求解器界面执行的，具体取决于所使用的求解器。

1.32. Qoca

Qoca是一个GPL库，用于逐步求解具有各种目标函数的线性方程组。它包含Cassowary的可靠实现，Cassowary是用于处理曼哈顿目标函数的流行线性规划算法。它用于多个免费软件项目中，并由Monash University维护。它具有C ++和Java版本，并提供Python和Perl的语言绑定。

1.33. SAS

SAS是由SAS Institute开发的一种统计软件套件，用于数据管理、高级分析、多元分析、商业智能、刑事调查和预测性分析。SAS是一个软件套件，可以挖掘、更改、管理和检索来自各种来源的数据，并对其进行统计分析。SAS为非技术用户提供了图形化的点击式用户界面，并通过SAS语言提供了更高级的选项。

SAS程序具有用于检索和处理数据的DATA步骤，以及用于分析数据的PROC步骤。DATA步骤具有使软件采取行动的可执行语句，以及提供读取数据集或更改数据外观的指令的声明性语句。DATA步骤分为两个阶段：编译和执行。在编译阶段，处理声明性语句并识别语法错误。之后，执行阶段按顺序处理每个可执行语句。PROC步骤由调用命名过程的PROC语句

组成。程序对数据集执行分析和报告，以生成统计信息、分析和图形。有300多个命名程序，每个程序都包含大量的规划和统计工作。PROC语句还可以显示结果，对数据进行排序或执行其他操作。

SAS的最大系列产品是其客户智能产品线。用于网络、社交媒体和市场分析的众多SAS模块可用于描述客户和潜在客户，预测其行为并管理和优化交流。SAS还提供了SAS欺诈框架。该框架的主要功能是监视不同应用程序、网络和合作伙伴之间的交易，并使用分析来识别表明欺诈的异常情况。SAS
Enterprise
GRC提供风险建模、场景分析和其他功能，以管理和可视化风险、合规性和公司政策。还有一个主要针对银行和金融服务组织而设计的SAS企业风险管理产品套件。SAS用于监视和管理IT系统操作的产品统称为SAS
IT管理解决方案。SAS从各种IT资产收集有关性能和利用率的数据，然后创建报告和分析。SAS的绩效管理产品在员工、部门和组织级别进行合并，并为关键绩效指标（KPI）提供图形显示。SAS供应链智能产品套件可满足供应链需求，例如预测产品需求、管理分销和库存以及优化定价。还有一套“
SAS for Sustainability
Management”软件，可以预测环境、社会和经济影响并确定运营之间的因果关系以及对环境或生态系统的影响。SAS具有针对特定行业的产品集，例如政府、零售、电信和航空航天，以及用于营销优化或高性能计算的产品集。

1.34. SCIP

SCIP是一个混合整数规划求解器，并且是Branch and cut和Branch and
price的框架，主要由柏林祖斯研究所开发。与大多数商业求解器不同，SCIP为用户提供了对求解过程的低级控制和信息。作为独立的求解器运行，它是混合整数程序最快的非商业求解器之一。

SCIP被实现为C可调用库。对于用户插件，提供了C ++ wrapper类。
LP松弛的求解器不是SCIP的本机组件，而是提供了开放的LP接口。当前支持的LP解算器是CLP、CPLEX、Gurobi、MOSEK、QSopt、SoPlex和Xpress。
SCIP可以在Linux、Mac、Sun和Windows操作系统上运行。

SCIP的设计基于约束的概念。它支持约20种约束类型，用于混合整数线性规划、混合整数非线性规划、混合整数全二次规划和伪布尔优化。它还可以解决Steiner树和多目标优化问题。SCIP有几个本机接口库，可以通过GAMS建模系统访问SCIP。在标准发行版中提供了与MATLAB和AMPL的接口。当前，还有两个用于Python和Java的外部接口。

1.35. SciPy

SciPy是一个免费的开源Python库，用于科学计算和技术计算。SciPy包含用于优化、线性代数、积分、插值、FFT、信号和图像处理、ODE求解器以及科学和工程中常见任务的模块。

SciPy建立在NumPy数组对象的基础上，并且是NumPy堆栈的一部分，该堆栈包括Matplotlib、pandas和SymPy等工具以及一组扩展的科学计算库。此NumPy堆栈具有与其他应用程序（例如MATLAB、GNU Octave和Scilab）相似的用户。
NumPy堆栈有时也称为SciPy堆栈。SciPy库当前是根据BSD许可证发行的，其开发由开发人员社区赞助和支持，也得到NumFOCUS的支持。SciPy关键算法和功能包是Python科学计算功能的核心。

1.36. SYMPHONY

网络上的单进程或多进程优化（SYMPHONY）是用于解决异构网络上的混合整数程序（MIP）的开源分支和剪切框架。它可以使用CLP、CPLEX、XPRESS或其他线性规划求解器来求解基础的线性规划。

SYMPHONY是一个可调用的库以解决MILP。库的用户可以通过提供专用于读取定制数据文件的特定于应用程序的子例程，生成特定于应用程序的切割平面或应用定制分支规则以多种方式来定制算法，从而生成定制化的分支与切割算法。该算法的大多数组件，例如搜索树管理、线性规划解决方案的管理、cut pool管理和通信管理，都在库内部，用户无需接触。可执行文件可以采用从完全顺序到完全并行的任何数量的配置来构建，并具有独立运行的切割生成器、cut pool和LP解算器。分布式版本当前在PVM消息传递协议支持的任何环境中运行，也可以使用任何符合OpenMP的编译器为共享内存体系结构编译相同的源代码。

SYMPHONY读取MPS（通过COIN-OR MPS阅读器）和GNU MathProg文件。SYMPHONY没有自己的LP解算器，但可以通过Osi接口与Clp、Cplex、Xpress等求解器一起使用。切割是使用COIN的切割生成库CGL生成的。SYMPHONY还针对旅行商问题、车辆路线问题、集合划分问题、混合邮递员问题等提供了特定于结构的实现。SYMPHONY还具有一个交互式shell，用户可以在其中输入命令来执行和控制程序。

2. 求解器比较

给定一个数学函数，描述如何将一组输入转换为输出。优化是指从一组可用方案中，生成和选择最佳解决方案。求解器系统地选择输入值，计算目标值，并记录在此过程中找到的最佳值。

在这个通用框架中可以模拟许多现实和理论问题。例如，输入是电动机的设计参数，输出是功耗；输入是业务选择，输出是获得的利润。通常，一个优化问题可以用下面的方式表示：

给定函数

求解：A中的元素x0，使得f（x0）≤f（x）（最小化问题）

通常，A是欧几里得空间的某些子集。最大化问题可以通过将函数乘以“-1”，转化为最小化问题。

使用求解器需要以合适的编程语言定义函数，求解器将在A中传递输入值，实现的软件模块将传递计算值f（x）。下表对主要的求解器进行了比较，这些求解器具有广泛的优化范围，包括专用库或通用库：

求解器	兼容的 编程语言	提供免费学术版	许可	备注
ALGLIB	C++, C#, FreePascal, VBA	Yes	Dual (Commercial, GPL)	通用库，包括优化包。
AMPL	C, C++, C#, Python, Java, Matlab, R	Yes	Dual (Commercial, academic)	一种流行的代数建模语言，用于线性、混合整数和非线性优化。学生和AMPL课程版本是免费提供的。
APMonitor	Fortran, C++, Python, Matlab, Julia	Yes	Dual (Commercial, academic)	用于混合整数和非线性优化的微分和代数建模语言。具有Matlab、Python和Julia的免费接口。
Artelys Knitro	C, C++, C#, Python, Java, Julia, Matlab, R	No	Commercial, Academic, Trial	通用库，专门用于非线性优化。处理混合整数问题（MINLP）和带有平衡约束的数学程序（MPEC）。非线性最小二乘问题的专用算法。
FICO Xpress	Mosel, BCL, C, C++, Java, R Python, Matlab, .Net, VB6	Yes	Commercial, academic, community, trial	优化技术和解决方案套件。包括：求解器(LP、MIP、MIQP、MIQCQP、MISOCP、MINLP QP、QCQP、SOCP、NLP)，代数建模和过程programming语言，集成开发环境，支持用于一系列执行服务，支持将优化模型和服务打包为软件解决方案。
GEKKO	Python	Yes	Dual (Commercial, academic)	GEKKO是一个Python软件包，用于机器学习以及混合整数和微分代数方程的优化。它与大规模求解器结合使用，可以进行线性、二次、非线性和混合整数规划（LP、QP、NLP、MILP、MINLP）。操作模式包括参数回归、数据协调、实时优化、动态仿真和非线性预测控制。

GNU Linear Programming Kit	C	Yes	GPL	免费的线性规划（LP）和混合整数规划（MIP）库。
GNU Scientific Library	C	Yes	GPL	GNU项目提供的免费库。
Gurobi	C, C++, C#, Java, .Net, Matlab, Python, R	Yes	Commercial, academic, trial	优化库。处理混合整数线性问题、凸二次约束和目标函数、多目标函数优化和SOS约束。
IMSL Numerical Libraries	C, Java, C#, Fortran, Python	No	Proprietary	
LIONsolver	C++, Java	Yes	Proprietary	根据RSO原则，支持交互式和学习优化。
Math Kernel Library	C++, Fortran	No	Proprietary	英特尔数字库。 MKL专门研究线性代数，但包含一些与优化相关的功能。
MIDACO	C++, C#, Python, Matlab, Octave, Fortran, R, Java, Excel, VBA, Julia	Yes	Dual (Commercial, academic)	用于单目标和多目标优化的轻量级软件工具，支持MINLP和并行化。
NAG Numerical Libraries	C, Fortran	No	Proprietary	
NMath	C#	No	Proprietary	基于MKL的C＃数值库。
Octeract Engine	C++/Python	No	Commercial	针对一般MINLP问题的超级计算确定性全局优化求解器。 Octeract Engine使用MPI进行分布式计算。
OptaPlanner	Java	Yes	ASL (open source)	Java中的轻量级优化求解器，带有用于JPA-Hibernate、Quarkus、Spring、Jackson、JAXB等的可选集成模块。在Kotlin和Scala上也适用。
SciPy	Python	Yes	BSD	Python的通用数值和科学计算库。

参考文献

https://en.wikipedia.org/wiki/Linear_programming

https://en.wikipedia.org/wiki/List_of_optimization_software

https://en.wikipedia.org/wiki/Comparison_of_optimization_software