最小费用流

这一章涵盖了“最小费用流”(Minimum Cost Flow)问题的相关内容。主要目标是让读者对这个问题有个大概的了解，所以并不包含大量的理论和实现的细节。如果您想深入地理解这个话题，可以阅读这一章末尾列出的相关书籍。在阅读本章内容之前，您首先需要了解图论中的一些概念，其中包括：最短路径，包含负权弧的路径，负权回路，最大流的相关概念以及解决最大流问题基本算法。

这一章包含了三节的内容，第一节介绍了最小费用流问题的定义和一些性质，第二节中给出了求解最小费用流问题的三个的算法，而最后一章中介绍了最小费用流在实际问题中的一些应用。

第一节　基本概念

问题描述：

为了说明什么是最小费用流问题，我们首先介绍一些重要的术语。我们用符号 $G=(V,E)$ 表示由**顶点(Vertex\Node)**集合 V 以及**边(Edge\Arcs)**集合 E 组成的有向网络。在这个有向网络中，每一条边 $(i,j)\in E$ 都有一个**容量(capacity)** u_{ij} ，表示能从这条边上通过的（液体，货物或其他可度量的东西的）最大的量。除此之外，还给每一条边 $(i,j)\in E$ 定义了一个**费用(cost)**值 c_{ij} ，表示要为通过这条边的每个单位的流量付出的代价。

我们还为每一个顶点 $i\in V$ 定义一个值 b_i ，用来描述这个顶点的供给/需求情况（也就是这个顶点能产生的在网络上流的东西的量减去这个顶点能够消费的量）。如果 $b_i>0$ ，就把顶点 i 称为**供给点(Supply Node)**；如果 $b_i<0$ 就把顶点 i 称为**需求点(Demand Node)**，注意需求点的需求量是 $-b_i$ 因为在实际问题中需求量是非负的；最后，把 $b_i=0$ 的顶点称为**中转点(Transshipment Node)**。

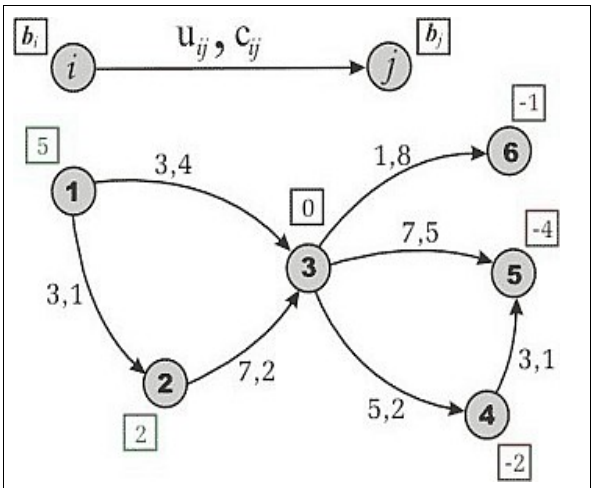

图 1 运输网络的一个实例，这里有两个供给点（供给量分别是5和2）三个需求点（需求量分别是1，4，2）还有一个中转点。每条边上都有两个值，用逗号隔开，表示这条边的容量和花费。

为了简化描述，我们把 G 称为一个**运输网络(Transportation Network)**，并且使用符号 $G=(V,E,u,c,b)$ 来明确指定网络中的每一个参数。

如果用符号 x_{ij} 来表示弧 $(i,j)\in E$ 上的流量，那么最小费用流问题的最优化模型可以表示为：

$$Minimize\ \ z(x)=\sum_{(i,j)\in E} c_{ij}x_{ij}$$

需要满足的约束条件为：

$$\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = b_i \quad for\ all\ i\in V,$$

$$0\leqslant x_{ij}\leqslant u_{ij} \quad for\ all\ (i,j)\in E$$

第一个约束条件的含义是：从一个顶点流出的总流量减去流入这个顶点的总流量应该等于这个顶点的供给/需求值（也就是说所有的顶点都没有积压，也没有消费过度）。这个约束条件被称为**流守恒性(Mass Balance Constrains)**。第二个约束条件，**容量限制(Flow Bound Constraints)**条件要求每一条弧上的流量不能超过该弧的容量，这个条件为每条弧上的流量规定了一个合法区间。用一个实际的例

子来说，上面的这个最优化模型可以用来描述仓库与商店之间的运货关系：假设我们只有一种产品，每个仓库有一定的库存量，每个商店也有自己的需求量，为了满足商店的进货需求，就需要把产品从若干个不同的仓库运到这些商店，要计算怎样运输能使运费最少，就是一个最小费用流的模型。如要求每个仓库都不能有产品积压，并且每个商店的需求都恰好得到满足，这就对应了流守恒性；如果每条运输线路能够运输的产品都是有限的，那么就有了容量限制的要求。

最小费用流问题可以用单纯形法来解决，但在这里我们只关注使用网络流理论来解决这类问题的方法。在我们开始介绍解决最小费用流问题的算法之前，首先复习一下所需要用到的理论知识。

寻找可行解：

怎样判断一个运输网络是否存在可行解呢？

如果 $\delta \overset{\text{def}}{=} \sum_{i \in V} b_i \neq 0$ ，那么问题没有可行解，因为在这样网络中，总的供给量和总需求量是不相等的，也就是说网络中一定出现剩余（ $\delta > 0$ ）或者过渡消费（ $\delta < 0$ ），流守恒条件无法成立。然而，我们可以很容易地把这样的网络转化成一个可以满足守恒条件的网络，方法是引入一个特殊顶点 r ，使它的供给/需求值等于 $-\delta$ 。然后，如果 $\delta > 0$ （供给过剩），那么就从所有的供给点($b_i > 0$)连一条容量为正无穷，费用为 0 的弧到这个特殊顶点；如果 $\delta < 0$ （需求过剩），我们就从引入的特殊顶点向所有的需求点($b_i < 0$)引一条容量为正无穷，费用为 0 的弧。显然，这个新的网络满足 $\sum_{i \in V \cup \{r\}} b_i = 0$ ，而且容易证明，在这个新的运输网络和原网络有相同的最优解，并且对应的目标函数有相同的最优值。

如果把引入的特殊顶点看作是垃圾场或是应急储备，那么构建的这个新的网络可以看作是这样一种情况：如果仓库储存的产品数量大于商店的需求，那么仓库就不得不把多余的产品当作垃圾（扔到垃圾场），这不需要运费；如果仓库存储的产品数量小于商店的需求，那么仓库不得不调用应急储备，并且假设这样也不用运费。虽然这样的情况可能不符合实际，但是却让我们很方便地把一个一般模型转化为了我们所定义的模型。注意，在这种情况下，我们很难说清楚最后求出的解对应的是一个什么样的现实问题，因为每个人都可以用不同的方法来解释引入的特殊顶点的含义。比如我们还可以假设进入特殊顶点的产品是那些因为过剩而留在仓库中的产品；把特殊顶点流出的产品看成是商店的没有被满足的需求等等。

即使 $\delta = 0$ 也不能保证弧的容量允许我们从供给点输送足够多的流量到需求点。如果给定网络存在可行流，我们希望找到一个能使所有的约束条件都得到满足运输方法。虽然我们找到的可能不是最优的，但它至少说明我们所面对的问题是存在可行解的。

接下来我们引入一个源点 s 和一个汇点 t 。对于每个顶点 $i \in V$ 且 $b_i > 0$ ，在网络 G 中增加一条弧 (s, i) ，它的容量为 b_i ，费用为 0；对于每个顶点 $i \in V$ 且 $b_i < 0$ ，在网络 G 中增加一条弧 (i, t) ，它的容量 $-b_i$ ，费用为 0。

这个新的网络被称作**转化网络 (Transformed Network)**。接下来，我们可以在转化网络中求从 s 到 t 的最大流，如果与源点汇点相关的弧全部被流填满，那么，原网络存在能够满足所有约束条件的可行解，否则不存在可行解。

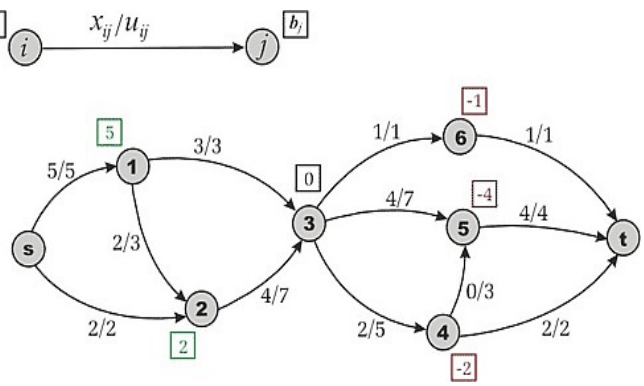

图 2 转化网络中的最大流，为了方便，我们省略了弧的花费。

假设原网络是存在可行解的，在求出最大流之后，如果我们把源点，汇点以及所有与之相关联的弧都去掉。我们该怎样判断现在网络中的流是不是最优的？它是否能使目标函数 z 取得最小值？我们暂时放下这些问题，先看一些假设条件。

是否每一个存在可行解的网络都存在最优解呢？如果在转化网络中从源点到汇点的某条路径上有容量为无限的负费用回路，那么目标函数的值是没有下界的。通常我们可以为容量无限的弧指定一个有限的容量来避免这种情况。

所以，从理论的观点来看，要解决最小费用流问题，我们必须首先验证以下几种情况：供给/需求是否平衡，是否存在可行解，是否存在容量无限的负费用回路。因为这些是问题存在可行解的必要条件。然而，在从实际的角度出发，我们可以一边寻找最优解，一边判断这些情况是否出现。

假设条件：

引入一些假设条件对于理解网络流问题的实质是很有帮助的，虽然有些假设可能会使问题丧失一般性。当然，即使没有这些假设条件，我们也可以解决网络流问题，只是会使问题的解变得非常复杂。

假设一 *问题中出现的所有数据* (u_{ij}, c_{ij}, b_i) *都是整数。*

由于我们可以给有理数乘上一个足够大的整数使之变成整数，所以在这个假设条件下，我们依然可以解决有理数范围内的网络流问题。

假设二 *网络是有向网络。*

如果是无向网络的，我们可以采取某些方法将它转换成一个有向网络。但是这只适用于无向边的费用非负的情况。

为了把一个无向网络转换成一个有向网络，我们可以将连接顶点 i 和 j 的无向边 (i,j) 用两条有向弧 (i,j) 和 (j,i) 代替，这两条弧的容量和费用都和它们所代替的无向边相同。首先说明对于原图中的每条边 $(i,j)\in E$ 都必须满足约束条件 $x_{ij} + x_{ji} \leqslant u_{ij}$ 并且这条边对目标函数的贡献为 $c_{ij}x_{ij} + c_{ij}x_{ji}$ 。可以证明，如果 $c_{ij} \geqslant 0$ ，那么肯定存在某个最优解，在这个最优解中 x_{ij} 和 X_{ji} 中至少有一个是0（如果 x_{ij} 和 X_{ji} 都大于0，那么可以在遵守流守恒性的条件下抵消一部分流量，由于 $C_{ij}\geqslant 0$ ，抵消后费用是降低的）。我们把这样的解叫做**不重叠(Non-Overlapping)**的解。容易看出，原网络中的每个不重叠的最优解都对应着一个转换后的网络中的最优解。那么为什么边权为负时不能这样转换，留给读者思考（提示：考虑正反流抵消后是不是还能保证费用降低以及约束条件 $x_{ij} + x_{ji} \leqslant u_{ij}$ 在新的网络中是否还能满足）。

假设三 *网络中的所有弧上的费用都是非负的。*

这个假设在一定程度上限制了问题的适用范围。接下来我们会说明当一个网络中存在负费用弧但不存在负费用回路时，可以把这个网络转化为一个不包含负费用弧的网络。对于存在负费用回路的情况，稍后我们会再介绍。

对于网络中的每个顶点 $i \in V$ 我们定义一个和该顶点相关联的值，把这个值称作该节点的**势(Potential)**，用符号 π 来表示。在确定了顶点的势之后，我们定义网络中弧 $(i,j) \in E$ 的**约减费用(Reduced Cost)** c_{ij}^{π} 为：

$$c_{ij}^{\pi} = c_{ij} + \pi_i - \pi_j$$

然后我们把每条弧上的费用 c_{ij} 修改为 c_{ij}^{π} ，看目标函数的值会发生怎样的变化。这里引入符号 $Z(x,\pi)$ 表示对于给定的 π ，修改费用后的目标函数值。显然，如果 $\pi = 0$ ，那么：

$$Z(x,0) = \sum_{(i,j)\in E} c_{ij}x_{ij} = Z(x)$$

如果 π 不全为0，可以得到如下结果：

$$
\begin{aligned}
z(x,\pi) &= \sum_{(i,j)\in E} c_{ij}^{\pi} x_{ij} = \sum_{(i,j)\in E} (c_{ij}+\pi_i-\pi_j)x_{ij} \\
&= z(x)+\sum_{(i,j)\in E}\pi_i x_{ij}-\sum_{(i,j)\in E}\pi_j x_{ij} \\
&= z(x)+\sum_{i\in V}\pi_i\sum_{\{j:(i,j)\in E\}}x_{ij}-\sum_{j\in V}\pi_j\sum_{\{i:(i,j)\in E\}}x_{ij} \\
&= z(x)+\sum_{i\in V}\pi_i\left(\sum_{\{j:(i,j)\in E\}}x_{ij}-\sum_{\{j:(j,i)\in E\}}x_{ji}\right) \\
&= z(x)+\sum_{i\in V}\pi_i b_i
\end{aligned}
$$

可以看出如果 π 是确定的，那么 $z(x,\pi)-z(x)$ 就是一个常数。所以，如果网络中的一个流能够使 $z(x,\pi)$ 取得最小值，那么这个流也能使 $z(x)$ 取得最小值，于是我们证明了：

定理一 *不论顶点的势 π 为何值，在容量相同的前提下，弧的费用为 c_{ij} 和 c_{ij}^{π} 的网络有相同的最优解，并且最优值满足关系：*

$$z(x,\pi)=z(x)+\sum_{i\in V}\pi_i b_i$$

下面的结论给出了约减费用的一个很重要的性质。

定理二 *令 G 表示一个运输网络，假定 P 是一条从顶点 $a\in V$ 到顶点 $b\in V$ 的有向路径，那么对于任何的 π ，都满足：*

$$\sum_{(i,j)\in P} c_{ij}^{\pi} = \sum_{(i,j)\in P} c_{ij}+\pi_a-\pi_b$$

假定 W 是一个有向环，那么对于任何的 π ，都满足：

$$\sum_{(i,j)\in W} c_{ij}^{\pi} = \sum_{(i,j)\in W} c_{ij}$$

根据这条定理，我们可以做如下推导：首先引入一个顶点 s ，然后从顶点 s 向所有其他顶点 $i\in V$ 引一条容量为一个正，费用为 0 的弧 (s,i) 。并且假设对于每一个顶点 $i\in V$ ， π_i 等于从顶点 s 到顶点 i 的最短路径长度（把弧上的费用 c 当作这条弧的长度，根据前面的假设，图中不存在负权回路，所以到所有顶点的最短路径都是存在的）。根据最短路径的性质，可以知道：

$$\pi_j \leqslant \pi_i + c_{ij}$$

所以：

$$c_{ij}^{\pi} = c_{ij}+\pi_i-\pi_j \geqslant 0$$

如果给定的运输网络中不存在负费用环，我们就可以引入一个顶点 s ，求出 s 到其他所有顶点的最短路经，然后把从 s 到网络中顶点的最短路径长度作为该顶点的势，修改弧的费用，这样我们就得到了一个不包含负费用弧的网络，并且根据上面的结论，这个网络中的最优解和原网络中的最优解是相同的（注意，最优解相同，但目标函数的最优值不一定相同）。要在包含负权边的图中求从源点到所有顶点的最短路径，可以采用 Bellman-Ford 算法。如果原网络中包含负费用回路，那么 $\pi_j \leqslant \pi_i + c_{ij}$ 这条性质就不能满足，而且根据定理二的第二条结论，必然存在某条边 $(i,j)\in E$ 的新费用 $c_{ij}^{\pi}<0$ ，也就不能通过上面的方法将网络中所有弧的费用转化为非负数。

假设四 *所有顶点的供给/需求情况服从条件 $\sum_{i\in V} b_i = 0$ ，网络中存在最小费用流的可行解。*

如果给定的网络不满足假设四的前半部分，我们或者说该网络不存在可行解，或者用前面介绍的方法将网络转化为一个满足条件的网络；如果给定的网络不满足假设的后半部分，那就是不存在可行解了，也就不用计算最优解了。

如果我们遇到的问题不满足上面的假设，我们可以把原始运输网络转化成满足上面这些假设的网络；然而在很多实际问题中，上面这些假设可能本来就是成立的，并不需要我们转换。

第二节 算法

残留网络

首先从最小费用流的角度考虑一下残留网络的概念。在最大流那一部分，您应该已经对这个概念有所了解，所以这里我们只是将这个的概念推广到最小费用流中去。

令 G 表示一个网络， x 表示该网络中的一个流。如果 E 中的弧 (i,j) 中输送了 x_{ij} 个单位的流，就定义弧 (i,j) 的残留容量为 $r_{ij}=u_{ij}-x_{ij}$ 。也就是说，我们还可以从顶点 i 向顶点 j 输送 r_{ij} 个单位的流，也可以取消已经存在的流 x_{ij} ，方法是在弧 (i,j) 上从顶点 j 向顶点 i 输送 r_{ij} 个单位的流。如过在弧 (i,j) 上增加一个单位的从 i 到 j 的流会使费用增加 c_{ij} ,而在这条弧上增加一个单位的从 j 到 i 的流将使费用减少 c_{ij} （取消了已经存在的流）。

沿着上面的思路，我们将给出残留网络的概念。在运输网络 $G=(V,E)$ 中，可以根据流 x 构造一个新的（残留）网络 $G_x=(V,E_x)$ ，其中 E_x 是一个弧集合，其中每条弧的容量是网络中存在流 x 时对应弧的残留容量。对于一个运输网络来说，残留网络和流是一一对应的。

那么 E_x 中到底包含了那些弧呢？这些弧的容量和费用又该如何定义呢？首先，我们要把原网络中的每条弧 (i,j) 拆成两条弧 (i,j) (j,i) ：其中的弧 (i,j) 的费用为 c_{ij} （残留）容量为 $r_{ij}=u_{ij}-x_{ij}$ ；弧 (j,i) 的费用为 $-c_{ij}$ （残留）容量为 $r_{ji}=x_{ij}$ 。然后我们在这些新的弧中选出残留容量大于 0 的放入到集合 E_x 中。

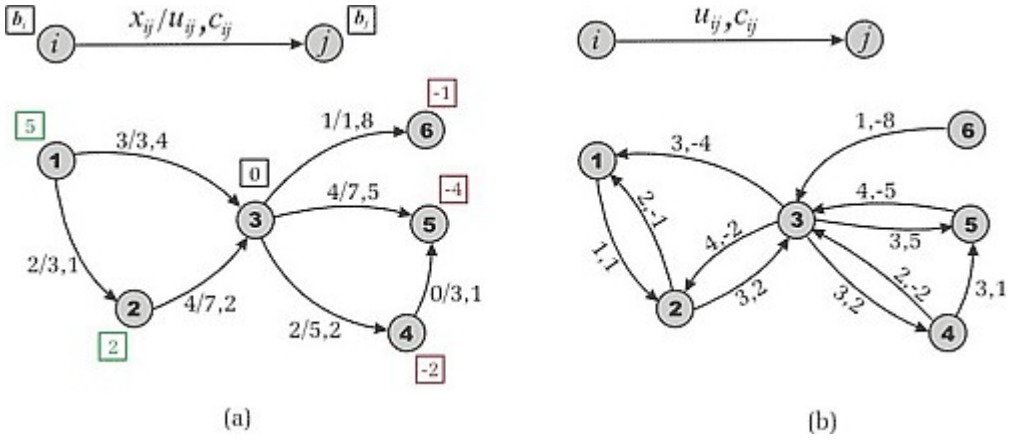

图 1 上一节中的运输网络的(a)一个可行流及(b)其对应的残留网络

要建立一个这样的残留网络在实现时可能会遇到一些困难，其中包括：

- 如果在 G 中同时包含了 (i,j) 和 (j,i) 这两条弧，那么在残留网络中，顶点 i 和顶点 j 之间可能存在 4 条弧（两条从 i 到 j ，两条从 j 到 i ），这样我们就不能使用邻接矩阵来保存了。为了避免这种情况发生，我们有两种选择：把原网络转化为一个任意两顶点之间只存在一条弧的网络，可以通过把每个顶点拆成多个顶点来实现；另外一种方法就是使用邻接链表，或者使用多个邻接矩阵。
- 如果在 G 中存在多条从 i 到 j 的弧，而且这些弧的费用不同，显然我们不能像计算最大流时那样简单地把这些弧的容量相加得到一条新的弧。所以我们必须想办法保存这些平行弧。

消圈算法

这一部分将介绍计算最小费用流的消圈算法，在这之前，先看两个重要的定理：

定理一：令 G 表示一个运输网络，如果 G 中不存在容量无穷的负费用回路，并且存在最小费用流的可行解，那么也存在最优解。

定理二：令 x 表示最小费用流问题的一个可行解，那么 x 是最有解当且进当残留网络 G_x 中不存在负费用回路。

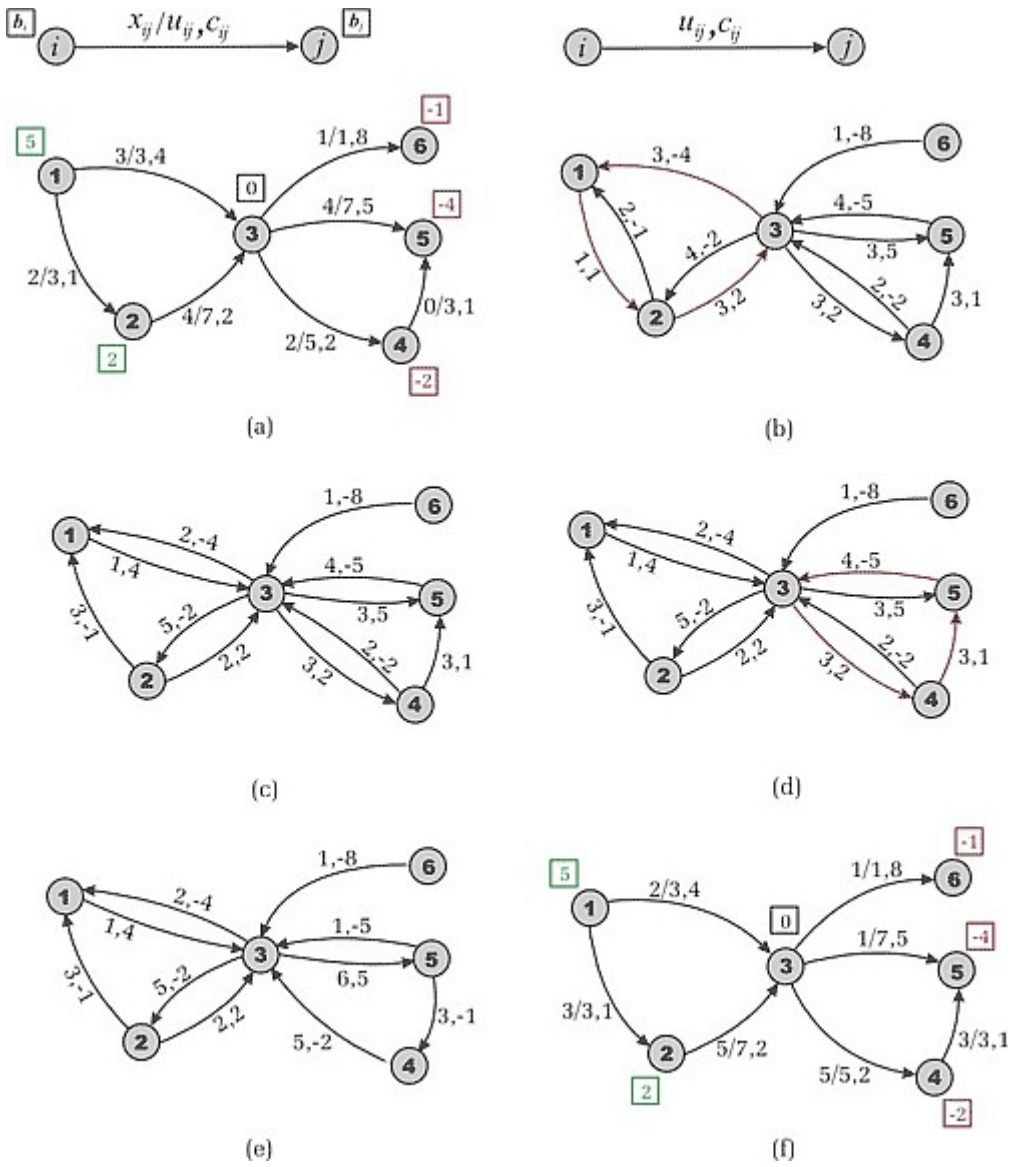

图 2 消圈算法 (a)计算一个可行解，费用为 54 。 (b)找到负费用回路 1-2-3-1，回路容量为 1，费用为－1。 (c)沿着负费用回路增大流量后的残留网络。 (d)找到另一个负费用回路 3-4-5-3，费用为-2 容量为 3。 (e)沿着负费用回路增大流量后的残留网络，此时不再包含负费用回路。(f)最有解已经求出，最小费用为 47。

定理二给出了计算最小费用流的消圈算法：首先计算网络的最大流得到一个可行解，然后通过在残留网络中寻找负费用回路并增大回路上的流量来减小花费。下面给出消圈算法的伪代码：

消圈算法
1 *寻找一个可行解* x
2 *While*(G_x *中存在负费用回路*) *do*
3 *找到负费用回路* W
4 $\delta \leftarrow min\{r_{ij}:(i,j)\in W\}$
5 *沿着负费用回路* W *增加* δ *个单位的流*
6 *更新* G_x

定理三：如果弧的容量，顶点的供给/需求都是整数，那么最小费用流一定存在整数解。

只要最小费用流存在最有解并且弧的容量，顶点的供给/需求都是整数，那么就可以用消圈算法

计算最小费用流。除此之外，不需要任何其他假设。

我们用 U 来表示弧的最大容量，用 C 来表示弧费用绝对值的最大值，用 m 表示网络 G 中弧的数目，用 n 表示顶点数目。那么对一个最小费用流问题来说，目标函数的绝对值不会超过 mCU 。每次消去负费用回路都会使目标函数严格地减小。根据我们的假设，所有的数据都是整数，那么 while 循环的执行次数为 $O(mCU)$ 。如果使用 Bellman-Ford 算法查找负费用回路，那么每次循环的复杂度为 $O(nm)$ 。所以，消圈算法计算最小费用流的复杂度为 $O(nm^2CU)$ 。

连续最短路算法

在消圈算法中，需要首先计算网络的最大流。而使用连续最短路算法，可以同时计算出网络的最大流和最小费用。

给定一个网路 G ，求这个网络的最小费用流。在前面的章节曾经提到，可以引入两个特殊顶点 s 和 t （源点和汇点），然后从 s 向所有 $b_i>0$ 的顶点 i 引一条容量为 b_i 费用为 0 的弧，从所有 $b_i<0$ 的顶点 i 引一条容量 $-b_i$ 费用为 0 的弧。然后，沿着从 s 到 t 的最短路（把弧的费用作为距离）增加流量，更新残留网络，再找从 s 到 t 的最短路...直到不存在从 s 到 t 的路径为止。这时的流已经是最大流，所以它是原网络的最小费用流的一个可行解，可以证明，它也是一个最有解。

只有在网络 G 中没有负费用回路时，才可以使用连续最短路算法。否则最短路是不存在的。接下来证明连续最短路算法的正确性。首先，假设原网络中没有负费用回路，那么沿着最短路径增大流量并不会使残留网络中出现负费用回路。所以，根据定理 2，当流 x 是最大流时，它也是最小费用流。

为什么沿着最短路径增大流量不会使残留网络中产生负费用回路呢？假设沿着最短路径 P 增大流量后，残留网络中出现了负费用回路 W ，而在此之前不存在负费用回路。那么一定是由于增大流量， P 上的某条弧 (i,j) （或某条子路径 i...j）的反向弧 (j,i) 出现，才产生负费用回路 W 的。而这个负费用回路的其他弧组成了一条从 i 到 j 的路径。由 W 的费用为负并且反向弧 (j,i) 的费用是 (i,j) 费用的相反数容易推出， W 上属于原网络的从 i 到 j 的路径的总费用比 P 上的从 i 到 j 的费用更小，这与 P 是最短路径矛盾，所以沿着最短路径增大流量不会使残留网络产生负费用回路。

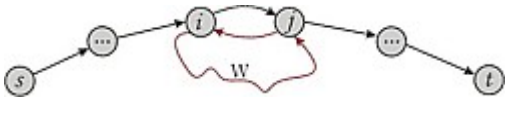

图 3 残留网络中的负费用回路的产生

如果边上的流量必须是整数，这个算法最多执行 $O(nB)$ 次增大流量的操作，这里的 B 表示每个顶点供给值的上限。如果我们使用 $O(nm)$ 算法寻找最短路径，那么连续最短路算法总的时间复杂度为 $O(n^2mB)$ 。

连续最短路算法

1 *在网络G中加入源点和汇点*
2 *将流x初始化为0*
3 *While*(G_x *中存在从* s *到* t *的路径*)*do*
4 *找一条从s到t的最短路径P*
5 *沿着路径P增大流量*
6 *更新残留流网络* G_x

回顾假设三中顶点的势的概念，我们可以通过 Bellman-Ford 算法计算从源点到其他顶点的最短路，然后把原网络转化为一个所有弧的费用均为非负的网络，并且这个新网络的最小费用流也是原网络的最小费用流。由于弧的费用都非负，我们可以在使用 Dijkstra 算法计算最短路从而提高效率。

为了达到这个目的，我们必须在每次计算最短路之后更新顶点的势和弧的费用，更新方法如下：

ReduceCost(π)
1 *Foreach*$(i,j)\in E_x$ *do*
2 $C_{ij}\leftarrow C_{ij}+\pi_i-\pi_j$
3 $C_{rev(i,j)}\leftarrow 0$

每找一次最短路之后我们都需要更新顶点的势，对于集合 V 中的每个顶点 i ，它的势就等于从 s 到 i 的最短路的长度。然后按照上面的方法更新弧的费用，可以发现更新后位于最短路径上的弧的费用会变成0，而其他的弧费用都是正的。这就解释了为什么我们要在上面过程的第3行把反向弧的费用改为0。因为我们是沿着最短路增加流量的，而最短路上的费用都是0。

因为在原网络中可能同时存在 (i,j) 和 (j,i) 这两条弧，所以我们引入符号 $rev(i,j)$ 来表示 (i,j) 的反向弧。下面是连续最短路算法的伪代码：

计算顶点势的连续最短路算法
1 *在网络G中引入源点汇点和需要的边*
2 *将网络中的流x初始化为0*
3 *使用Bellman−Ford算法计算顶点的势*π
4 *Reduce Cost*(π)
5 *while*(G_x*中存在从s到t的路径*) *do*
6 *找一条从s到t的最短路P*
7 *ReduceCost*(π)
8 *沿着路径P改进当前流x*
9 *更新残留网络*G_x

在进入第5行的循环之前，必须首先计算顶点的势 π 并且使弧上的容量非负，以后每次改进流 x 之前也必须做同样的工作。由于费用非负，在第6行可以使用Dijkstra算法计算最短路。第8行改进当前流 x 的操作过后弧的费用仍然都是非负数，所以在下一次迭代过程中， Dijkstra算法仍然适用。

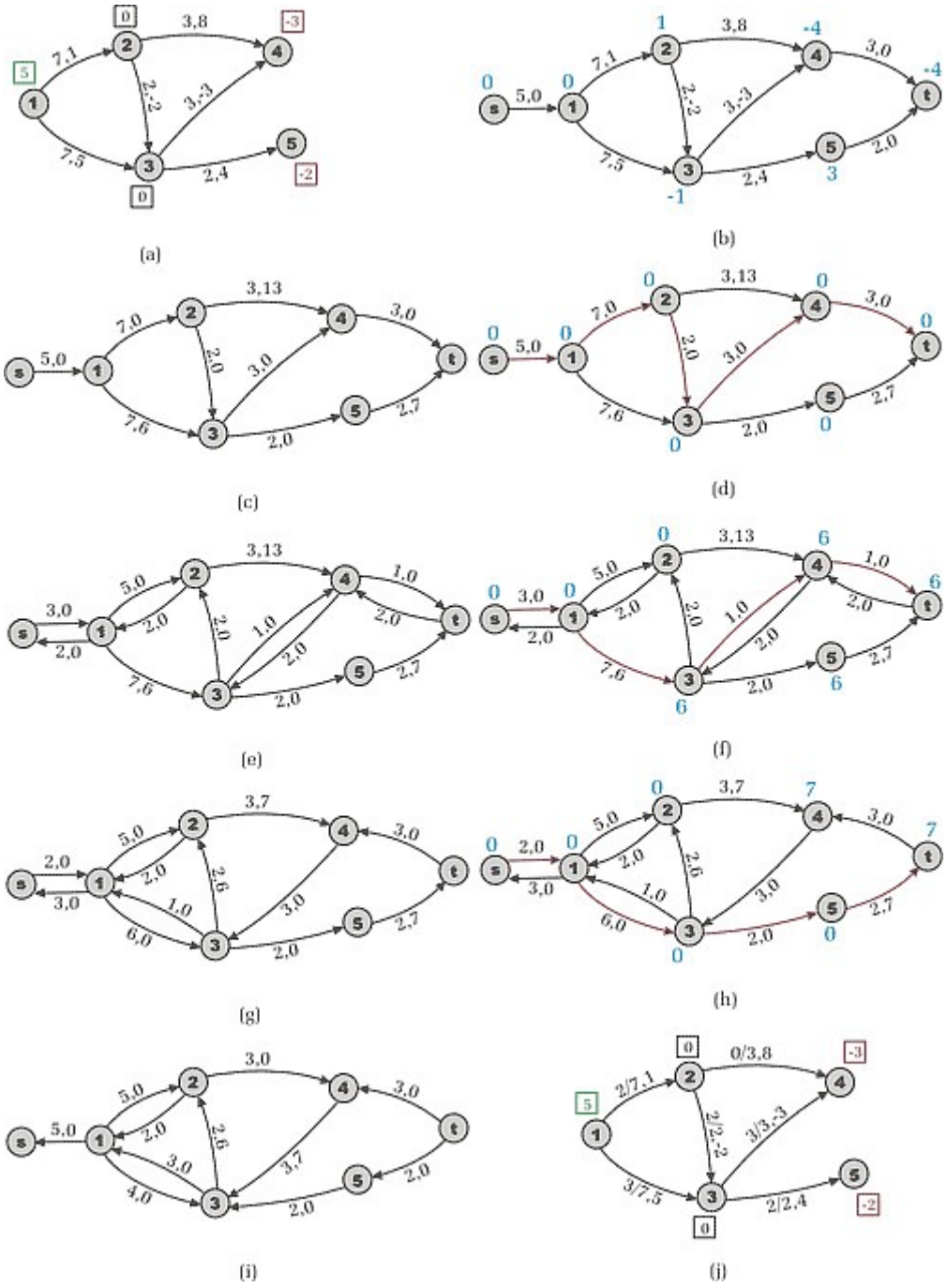

图 4 连续最短路算法：(a)初始化(b)执行第 3 行计算出的顶点的势(c)第 4 行更新弧的费用(d)找到一条容量为 2 的增广路 s-1-2-3-4-t 并且计算新的顶点的势(e)更新弧的费用后的残留网络(f)找到一条容量为 1 的增广路 s-1-3-4-t(g) 更新弧的费用后的残留网络(h)找到一条增广路 s-1-3-5-t 并且重新计算计算顶点的势(i)残留网络中不包含增广路(j)把求得的流放到原网络中去，计算得最大流的最小费用是 12。

我们只使用了一次 Bellman-Ford 算法来避免网络中出现负费用弧，这个过程的时间复杂度为 $O(nm)$ 。然后，需要使用 $O(nB)$ 次 Dijkstra 算法（复杂度为 $O(n^2)$ 或 $O(mlogn)$ ）。那么连续最短路算法计算最小费用流的时间复杂度就是 $O(n^3B)$ 或 $O(mnBlogn)$ 。如果在计算最短路的过程中使用 Fibonacci 堆，那么计算一次最短路的复杂度为 $O(nlogn+m)$ 。但在实际使用时的效果并不好。