第 10 章 近似算法

许多实际应用问题都是 *NP*-完全问题，这类问题很可能不存在的多项式时间算法。一般而言，*NP*-完全问题可采用如下三种方式处理。如果问题的输入规模较小，则可以利用第 8 章的搜索策略在指数时间内求解问题。如果输入规模较大，则既可以利用第 9 章的随机算法在多项式时间内“高概率”地精确求解问题，也可以考虑在多项式时间内求得问题的一个“近似解”。**近似算法**是指能够在多项式时间内给出优化问题的近似优化解的算法，近似算法不仅可用于近似求解 *NP*-完全问题，也可用于近似求解复杂性较高的 *P* 问题。由于 *NP*-完全问题的广泛性和计算规模的爆炸性增长，学习和掌握近似算法设计的主要方法变得日益重要。本章将介绍近似算法的相关概念和设计近似算法的主要方法。

本章约定所讨论的计算问题均是优化问题，即在给定的约束下求出使得代价函数达到最大（或最小）的可行解计算问题，其中可行解指的是满足给定约束条件的解。每个可行解均可以由代价函数计算得到一个正的代价。问题的优化解是使得代价函数达到最大（或最小）的可行解。求代价函数达到最大(小)值的问题称为最大（小）化问题。

若无特别说明，本章涉及的问题均是 *NP*-完全问题。

10.1 近似算法的性能分析

近似算法的性能分析包括时间复杂度分析，空间复杂度分析和近似精度分析，其中时间（空间）复杂度的分析同精确算法的分析。近似精度分析是近似算法特有的，它主要用于刻画近似算法给出的近似解相比于问题优化解的优劣程度。目前，存在三种刻画近似精度的度量，即近似比，相对误差界和 $1+\varepsilon$-近似。下面，先介绍它们的概念，然后指明它们之间的关系。注意，这三种度量方式在文献中均很常见。

定义 10.1(近似比). 称优化问题的一个近似算法 A 的近似比为 $p(n)$，如果 $max\{c/c^*, c^*/c\}\le p(n)$对问题的所有输入规模为 n 的实例成立, 其中 c 是近似算法所产生的近似解的代价，c^*是问题实例的优化解代价，$p(n)$是 n 的函数。

性质 10.1.近似算法的近似比 $p(n)\ge 1$ 恒成立。

证明.对于最大化问题，$c\le c^*$且 $max\{c/c^*, c^*/c\}=c^*/c$; 对最小化问题 $c\ge c^*$且 $max\{c/c^*, c^*/c\}=c/c^*$。注意，$c/c^*<1$ 当且仅当 $c^*/c>1$。 □

求解优化解的精确算法的近似比是 1。近似算法的近似比反映了近似算法所求近似解的精度。近似比越大，近似解距离优化解的距离越大。如果一个近似算法的近似比 $p(n)$为常数，则称该近似算法具有常数近似比。

定义 10.2 (相对误差)一个近似算法在任意实例上的相对误差定义为$|c-c^*|/c^*$, 其中 c 是近似算法所产生的近似解的代价，c^*是问题实例的优化解的代价。

定义 10.3(相对误差界) 称一个近似算法的相对误差界为 $\varepsilon(n)$，如果算法的在规模为 n 的任意实例上的相对误差满足$|c-c^*|/c^*\le\varepsilon(n)$，其中 $\varepsilon(n)$是 n 的函数。

显然，相对误差界总是非负的。

性质 10.2. $\varepsilon(n)\le p(n)-1$。

证明. 对最小化问题，有 $\varepsilon(n)=|c-c^*|/c^*=(c-c^*)/c^*=c/c^*-1=p(n)-1$。

对最大化问题有 $\varepsilon(n)=|c-c^*|/c^*=(c^*-c)/c^*=(c^*/c-1)/(c^*/c)=(p(n)-1)/p(n)\le p(n)-1$。 □

对于某些问题的近似算法，$\varepsilon(n)$和 $p(n)$独立于输入规模 n，则用 p 和 ε 表示之；通常认为，这类近似算法具有良好的性能。另有一些问题，很难找到具有常数近似比的多项式时间近似算法，此时近似比的常见形式为 $\log n$，$\sqrt{n}$ 等。这类近似算法的性能则通常被认为比较差。当然，这都是相对的。通常认为，近似比越小（或者近似比的阶越低），则近似算法性能越好。

定义 10.4(1+ε-近似算法) 优化问题的一个近似算法 A，如果其相对误差界为 $\varepsilon(n)$，则称之为一个 $1+\varepsilon(n)$-近似算法。

根据性质 10.2，$1+\varepsilon$-近似算法既指明了算法的相对误差界限又指明了算法的近似比。

设计近似算法时，如果期望的近似解精度越高（或越低），算法的运行时间将会越长（或越短）。下面的完全多项式近似模式精确定义了这样的算法族。

定义 10.5(近似模式) 如果优化问题的近似算法 $A(I, \varepsilon)$在输入的问题实例 I 和相对误差界限 ε 上输出相对误差不超过 ε 的近似解，则称该近似算法为一个近似模式。

近似模式表示了一族近似算法，其中每个算法的近似度各不相同，因而时间复杂度也不相同。当 ε 固定时，近似模式的复杂度是$|I|$的多项式。这个多项式会随着 ε 的变化而变化。因此，近似模式的复杂度是$|I|$和 ε 的函数。例如近似模式的时间复杂度 $T(|I|,\varepsilon)$形如 $O(|I|^2/\varepsilon^3)$或 $O(|I|^{1/\varepsilon})$。一般来说，相对误差界限 ε 越小，算法的复杂度越高。但时间复杂度随 $1/\varepsilon$ 指数级增大仍是不能接受的。

定义 10.6 (多项式近似模式)如果近似模式 $A(I, \varepsilon)$的时间复杂度是$|I|$的多项式，则称之为一个多项式时间近似模式。

定义 10.7 (完全多项式近似模式)如果近似模式$A(I, \varepsilon)$的时间复杂度是$|I|$和 $1/\varepsilon$的多项式，则称之为一个完全多项式时间近似模式。

例如，如果近似模式 $A(I, \varepsilon)$的运行时间是$|I|^2/\varepsilon^3$，则 $A(I, \varepsilon)$是一个完全多项式时间近似模式。

10.2 基于组合优化的近似算法

本节通过一系列例子介绍一般的组合优化策略在近似算法设计中的应用。

10.2.1 顶点覆盖问题的近似算法

顶点覆盖问题（Vertex-Cover Problem）的输入是一个图 $G=(V,E)$,输出是顶点数最少的子集 $C\subseteq V$ 使得 E 中的每条边均被 C 覆盖（即至少有一个端点位于 C 中）。顶点覆盖问题可如下近似求解。先将 C 初始化为空集；然后，重复下列操作直到 E 中所有边均被 C 覆盖：选择一条未被 C 覆盖的边(u,v),将两个端点 u 和 v 加入 C 中；并删除所有已被 u 或 v 覆盖的边。算法的形式化描述如下。

顶点覆盖近似算法 APPROX-Vertex-Cover

1.$C\leftarrow\varnothing$, $E'\leftarrow E$
2.while $E'\neq\varnothing$ DO
3. 任取$(u,v)\in E'$;
4. $C\leftarrow C\cup\{u,v\}$

5. $E'\leftarrow E'-\{(u',v')|(u',v')\in E, u'=u$ 或 $v'=v\}$

6.输出 C

显然，算法 ApproxVertexCover 的时间复杂度为$\Theta(|E|)$。

定理 10.3. 算法 ApproxVertexCover 的近似比为 2。

证明: 令 $A=\{(u,v)|\ (u,v)$是算法第 3 步选中的边$\}$。若$(u,v)\in A$，则算法将所有与(u,v)邻接的边皆从 E'中删除。因此，A 中无邻接边。又由于算法第 4 步中，对添加到 A 每条边(u,v)，将它的两个端点同时添加到输出集合 C，故$|C|=2|A|$。

设 C^*是问题的优化解，则 C^*必然覆盖 A 中的每条边。由于 A 中无邻接边，故 C^*至少包含 A 中每条边的一个端点。于是，$|A|\le|C^*|$。

由此可得，$|C|=2|A|\le 2|C^*|$，即$|C|/|C^*|\le 2$。 □

10.2.2 装箱问题的近似算法

装箱问题(Bin-Packing Problem)的输入是体积分别为 $a_1,a_2,\ldots,a_n\in(0,1]$的 n 个物品和无穷个容积为 1 的箱子,输出 n 个物品的一个装箱方案使得所用箱子个数最少。

例如，在实例 0.3,0.5,0.8,0.2,0.4 上的最优装箱方案是{0.2,0.8},{0.3,0.5},{0.4}，它使用 3 个箱子。

装箱问题是许多实际问题的抽象。例如，考虑将 n 张尺寸不同的票卷印刷在最少的具有标准尺寸的纸张上。这是一个特殊的装箱问题，其中箱子对应具有标准尺寸的纸张，每张票券对应一个物品，可通过标准化确保箱子的体积为 1。

装箱问题可以用如下简单策略给出一个近似解。顺序地将每个物品装箱，设前 i-1 个物品 $a_1,a_2,...,a_{i-1}$ 已经放入箱子 $B_1,...,B_k$ 中，考虑第 i 个物品的装箱策略。依次考察箱子$B_1,...,B_k$，找到第一个能够容纳物品 i 的箱子B_j，将物品 i 放入B_j；如果$B_1,...,B_k$ 中不存在能够容纳物品 i 的箱子，则开启新箱子 B_{k+1} 并将物品 i 放入 B_{k+1}。重复上述过程直到所有物品被装箱。算法的形式化描述如下。

装箱近似算法 FirstFit

1.$k\leftarrow 1$, $B_1\leftarrow\varnothing$;

2.For $i\leftarrow 1$ TO n DO

3. 从 $B_1,B_2...,B_k$ 中选择第一个能容纳 a_i 的箱子 B_j, 令 $B_j\leftarrow B_j\cup\{a_i\}$;

4. 若 B_j 不存在，则令 $B_{k+1}\leftarrow\{a_i\}$，$k\leftarrow k+1$;

5. 输出 $B_1,B_2,...,B_k$;

算法 FirstFit 的时间复杂度为 $O(n^2)$。这是由于，算法处理第 i 个物品时，最坏情况下需要检查 $B_1,B_2...,B_k$ 中的每个箱子，而此时 $k\le i-1$。故 $T(n)\le\sum_{1\le i\le n}(i-1)=O(n^2)$。

下面分析算法的近似比。设 k 和 k^*分别表示算法 FirstFit 给出的近似解和问题的优化解中所用箱子的个数。显然，$k^*\ge\sum_{1\le i\le n}a_i$，这是由于 n 个物品的总体积为$\sum_{1\le i\le n}a_i$ 且每个箱子具有单位容量。另一方面，将近似解 $B_1,B_2,...,B_k$ 中各个箱子内物品的总体积分别记为$|B_1|,|B_2|,...,|B_k|$，则$|B_1|+|B_2|+...+|B_k|=\sum_{1\le i\le n}a_i$。注意到，$|B_i|+|B_j|>1$ 对近似解中任意两个箱子 B_i 和 B_j 成立；否则，FirstFit 算法会将 B_j（不妨设 $j>i$）中的物品放入 B_i 中。于是，

$$k/2<(|B_1|+|B_2|)/2+(|B_2|+|B_3|)/2+...+(|B_{k-1}|+|B_k|)/2+(|B_k|+|B_1|)/2$$
$$=|B_1|+|B_2|+...+|B_k|$$

$=\sum_{1\le i\le n}a_i$

$\le k^*$

定理 10.4. 算法 FirstFit 的近似比为 2。 □

10.2.3 最短并行调度问题的近似算法

最短并行调度问题的输入是 m 台完全一样的机器和运行时间分别 $t_1,...,t_n$ 的任务，每个任务均需要排他地在一台机器上运行，问题输出是 n 个任务在 m 台机器上的一个调度使得这些任务的并行时间最短，其中并行时间指的是所有任务结束处理的最晚时间。

一个简单的调度策略如下。首先，任意排定所有计算任务的一个顺序，然后顺序将每个任务分配到 m 台机器中的一台。设前 i-1 项任务分配后各台机器上任务的总时间分别为 $T_1,...,T_m$，则将第 i 项任务分配给总时间最短的机器 j，即 $T_j=min_{1\le k\le m}T_k$。

最短并行调度算法 MinMakespanScheduling

输入： m 台完全一样的机器，运行时间分别 $t_1,...,t_n$ 的任务

输出： 输入任务在 m 台机器上的一个调度，使得平行时间最短

1.任意排定所有任务的一个顺序 $t_1,...,t_n$;

2.$T_k\leftarrow 0$; $M_k\leftarrow\varnothing$; 其中 k=1,2,...,m

3.For $i\leftarrow 1$ To n Do

4. 找出 j 使得 $T_j=min_{1\le k\le m}T_k$;

5. $T_j\leftarrow T_j+t_i$; $M_j\leftarrow M_j\cup\{i\}$;

6.输出 $M_1,M_2,...,M_m$

显然，最短并行调度算法的时间复杂度为 $\Theta(n)$。

下面分析算法的近似比，分别用 T^*和 T 表示问题优化解和最短并行算法给出的近似解的并行时间。显然，$T^*\ge (\sum_{i=1}^{n}t_i)/m$,这是由于 n 项任务在 m 台机器至少平均需要时间$(\sum_{i=1}^{n}t_i)/m$,从而结束时间最晚的机器至少耗时$(\sum_{i=1}^{n}t_i)/m$。同时，$T^*\ge t_i$ 对任意 $1\le i\le n$ 成立，因为每个任务均需要排他地在一台机器上处理。

另一方面，设在算法输出的近似解中，第 j 台机器的处理时间最长（如图 10.1 所示）且第 j 台机器上最后处理的任务是第 h 个任务，将第 h 个任务在第 j 台机器开始处理的时间记为 T_{start}。易知,$T=T_{start}+t_h$ 且 $T_{start}\le(\sum_{i=1}^{n}t_i)/m\le T^*$。于是，

$T=T_{start}+t_h\le T^*+T^*=2T^*$。

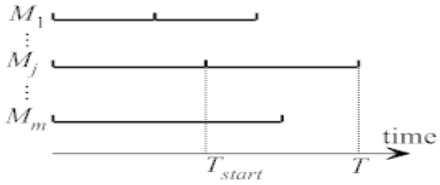


图 10.1 最短并行调度的近似解

定理 10.5. 算法 **MinMakespanScheduling** 的近似比为 2。 □

10.2.4 旅行商问题的近似算法

本小节给出如下定义的旅行商问题的一个 2-近似算法，习题 10.3 要求用类似的思想设计一个 3/2-近似算法。

满足三角不等式的旅行商问题

输入：加权的完全无向图 $G=(V,E)$, 权值函数 c：$E\to \boldsymbol{R}^+$，c 满足三角不等式

输出：权值最小的哈密顿环

哈密顿环指的是包含 V 中每个顶点恰一次的简单环。哈密顿环的权值指的是其中所有边的权值之和。如果旅行商问题的权值函数满足三角不等式，即 $c(u,w)\le c(u,v)+c(v,w)$对$\forall u,v,w\in V$ 成立，则称它满足三角不等式。无论旅行商问题是否满足三角不等式，它均是 *NP*-完全问题。10.7.1 节将证明，不满足三角不等式的旅行商问题不存在常数近似比的近似算法，除非 $NP=P$。

求解旅行商问题要求选用图中的一组边，将图中所有顶点连接成一个简单环。在图中，最小生成树 T^*以总代价最小的方式将所有顶点相互连接。因此，以深度优先方式遍历最小生成树 T^*中的每条边两遍，可得到访问所有顶点的回路 L，然后再将回路 L 改造成一个简单环 C，从而得到问题的一个近似解 C。分析近似比时，问题的优化解 C^*和近似解 C 可以通过最小生成树 T^*关联。由此，得到如下算法。

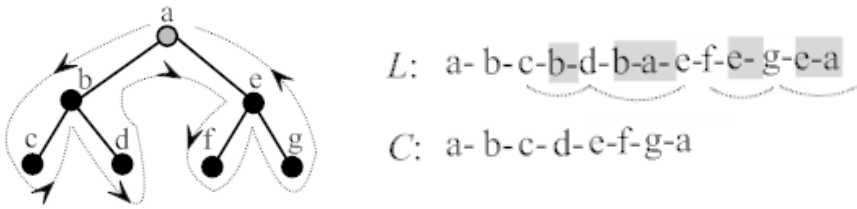
旅行商问题近似算法 ApproxTSPTour

输入：加权的完全无向图 $G=(V,E)$, 权值函数 c：$E\to \boldsymbol{R}^+$，c 满足三角不等式

输出：权值最小的哈密顿环

1. 任意选择 $r\in V$ 作为生成树的根；
2. 调用 Prim 算法（参见第 5 章）得到图 $G=(V,E)$的一棵最小生成树 T^*；
3. 先根遍历 T^*，访问 T^*中每条边两遍，得到顶点序列 L；
4. 删除 L 中的重复顶点形成哈密顿环 C;
5. 输出 C

例如，图 10.2 示意了算法 ApproxTSPTour 的计算过程。



(a)最小生成树 T^*的遍历　　(b)由遍历结果 L 构造近似解 C

图 10.2 旅行商问题近似算法操作过程示意

算法 ApproxTSPTour 的时间复杂度为 $O(|V|^2\log|V|)$。事实上，第 2 步的开销为 $O(|E|\log|V|)$且图 G 是完全图。第 3-4 步的开销为 $O(|V|)$,因为最小生成树恰有$|V|$-1 条边。

下面分析算法的近似比，用 C^*和 C 分别表示问题的优化解和算法输出的近似解。一方面，C^*是包含所有顶点的简单环，删除其中一条边得到图 G 的一棵生成树 T。由于 T^*是图 G 的最小生成树，因此，$c(C^*)\ge c(T)\ge c(T^*)$。

另一方面，L 是遍历 T^*的每条边两次得到的回路，故 $c(L)=2\cdot c(T^*)$。删除 L 中重复顶点得到近似解 C，且删除 L 中的每个重复顶点相当于用一个三角形的第三边替换其余两边，如图 10.2(b)所示。在满足三角不等式的旅行商问题中，$c(C)\le c(L)=2\cdot c(T^*)$。

联立上述两段得到的不等式，有 $c(C)\le c(L)=2\cdot c(T^*)\le 2\cdot c(C^*)$。

定理 10.6. Approx-TSP-Tour 算法的近似比为 2。 □

10.2.5 子集和问题的完全多项式近似模式

本小节讨论如下定义的子集和问题。我们首先给出求解子集和问题优化解的指数时间算法，然后将它改造成一个完全多项式近似模式。

子集和问题

输入：有限正整数集合 $S=\{x_1,x_2,...,x_n\}$和正整数 t

输出： $\sum_{x\in A}x$，其中 $A\subseteq S$，$\sum_{x\in A}x\le t$ 且 $\sum_{x\in A}x=max\{\sum_{x\in B}x \mid B\subseteq S,\sum_{x\in B}x\le t\}$

指数时间算法

为了求解问题的优化解，我们用系统化方法枚举 S 的所有子集和。枚举方法的基本思想是，依次对 i=0,1,...,n-1，利用$\{x_1,x_2,...,x_i\}$的所有子集和构造$\{x_1,x_2,...,x_i,x_{i+1}\}$的所有子集和，最终得到 S 的所有子集和。考虑对于$\{x_1,x_2,...,x_i,x_{i+1}\}$的任意子集 A, 如果 $x_{i+1}\notin A$，则 A 中元素之和即为$\{x_1,x_2,...,x_i\}$的一个子集和；否则，A 中元素之和是 x_{i+1} 加上$\{x_1,x_2,...,x_i\}$的一个子集和。因此，为了完成计算，只需将$\{x_1,x_2,...,x_i\}$的所有子集和 L_i 与 L_i+x_{i+1} 合并即可,其中 $L_i+x_{i+1}=\{y+x_{i+1}\mid y\in L_i\}$。

为了操作方便，将 L_i 维护成一个有序列表。于是，L_i+x_{i+1} 也是一个有序列表。合并 L_i 和 L_i+x_{i+1} 通过调用过程 MergeList(L_{i-1},$L_{i-1}+x_i$)来完成。请读者给出 MergeList 的一个实现，确保其时间复杂度为 $O(|L_i|)$。

子集和算法 ExactSubsetSum(*S*,*t*)

输入：有限正实数集合 $S=\{x_1,x_2,...,x_n\}$和正实数 t

输出： $\sum_{x\in A}x$，其中 $A\subseteq S$，$\sum_{x\in A}x\le t$ 且 $\sum_{x\in A}x=max\{\sum_{x\in B}x\le t \mid B\subseteq S\}$

1. $n\leftarrow|S|$
2. $L_0\leftarrow$<0>
3. For $i\leftarrow$1 To n Do
4. $L_i\leftarrow$MergeList(L_{i-1},$L_{i-1}+x_i$)；
5. 从 L_i 中删除大于 t 的元素；
6. 输出 L_n 中最大值 z；

图 10.3 给出了 ExactSubsetSum 算法在$\{x_1,x_2,x_3\}$上的计算过程。

L_0=<0> L_0+x_1=<x_1>

L_1=<0,x_1> L_1+x_2=<x_2,x_1+x_2>

L_2=<0,x_1,x_2,x_1+x_2> L_2+x_3=<x_3,x_1+x_3, x_2+x_3, $x_1+x_2+x_3$>

L_3=<0,x_1,x_2,x_1+x_2,x_3,x_1+x_3,x_2+x_3,$x_1+x_2+x_3$>

图 10.3 ExactSubsetSum 算法在$\{x_1,x_2,x_3\}$上的计算过程示意

利用数学归纳法，不难证明 L_i 枚举了$\{x_1,x_2,...,x_i\}$小于等于 t 的所有子集和。因此，L_n 枚举了 S 小于等于 t 的所有子集和。故，ExactSubsetSum 算法是正确的。

下面分析算法的时间复杂度 $T(n)$。同样，利用数学归纳法易证$\cdot|L_i|\le 2^i$对 $0\le i\le n$ 成立，并且在 t 充分大时算法第 5 步不能删除任何元素，此时$|L_i|=2^i$。由此可知，当循

环变量为 i 时，第 4 步的开销为$\Theta(2^i)$。因此，

$$T(n)=\Theta(1+2^1+2^2+...+2^n)= \Theta(2^n)。$$

有序列表的修剪算法

不难发现，算法 ExactSubsetSum 的时间复杂度很高的原因在于，最坏情况下，$|L_{i+1}|=2|L_i|$。亦即，算法维护的有序列表的长度成倍增长。能否通过删减 L_i 中部分元素，确保算法在多项式时间内求得问题的一个近似解呢？答案是肯定的。在此，先讨论有序列表的修剪，后面再给出近似算法。

修剪有序列表 L 的目标有两个。一是删除尽可能多的元素，二是控制近似精度。具体地讲，给定有序列表 L 和修剪参数 $\delta\in(0,1)$。 根据 δ 修剪 L 是指：(1)从 L 中删除尽可能多的元素；(2)如果 L'是修剪 L 后的结果，则对每个从 L 中删除的元素 y，存在 $z\in L'$使得$(y-z)/y\le\delta$ 或$(1-\delta)y\le z\le y$，亦即 z 相对于 y 的相对误差小于 δ。

实现上述修剪目标可采用贪心策略。保留 L 中第一个元素 $y_1=last$；然后，依次扫描 L 每个元素，删除与 $last$ 的相对误差小于等于 δ 的所有元素，保留首个与 $last$ 的相对误差大于 δ 的元素 y_j 并更新 $last$ 为 y_j。实现该过程的算法如下。

修剪算法 Tim(L, δ)
输入：修剪参数 $\delta\in(0,1)$和有序列表 $L=\langle y_1,y_2,...,y_m\rangle$,其中 $y_i\le y_{i+1}$;
输出：修剪后的有序列表 L'
1.$m\leftarrow|L|$;
2.$L'\leftarrow<y_1>$，$last\leftarrow y_1$; /*$last$ 记录最近添加到 L'的元素*/
3.For $i\leftarrow2$ To m Do
4. If $last<(1-\delta)y_i$ Then /* $last$ 与 y_i 的相对误差大于 δ */
5. 将 y_i 添加到 L'末尾
6. $last\leftarrow y_i$
7.输出 L'

例如，给定 δ=0.1 和 L=<10,11,12,15,20,21,22,23,24,29>, 则 Trim 算法修剪 L 的结果为 L'=<10,12,15,20,23,29>。

Trim 算法的时间复杂度为$\Theta(m)$,其中 $m=|L|$。事实上，Trim 算法第 3-6 步的循环扫描 L 中的每个元素，且 4-6 步的每一步的开销均为$\Theta(1)$。

完全多项式近似模式

给定相对误差界限ε和子集和问题的输入。借助子集和算法框架，求得 L_i 之后，利用ε/n 最为修剪参数调用 Trim 算法修剪 L_i。最后，L_n 中的最大元素给出问题的一个近似解。

子集和问题的完全多项式近似模式 ApproxSubsetSum(S,t,ε)
输入：$S=\{x_1,x_2,...,x_n\}$, $t\ge0$, $0<\varepsilon<1$
输出：子集和问题的一个近似解
1. $n\leftarrow|S|$
2.$L_0\leftarrow<0>$
3.For $i\leftarrow1$ To n Do
4. $L_i\leftarrow$Merge-List(L_{i-1},$L_{i-1}+x_i$)

5. $L_i \leftarrow \mathrm{Trim}(L_i, \varepsilon/n)$ /* 修剪参数 $\delta=\varepsilon/n$ */
6. 从 L_i 中删除大于 t 的元素
7.输出 L_n 中最大值 z

为便于后面的讨论，令

$$P_i = P_{i-1} \cup P_{i-1}+x_i$$
$$=\{x \mid x=\sum_{y\in A} y, A\subseteq\{x_1,x_2,...x_i\}\}$$

是$\{x_1,x_2,...x_i\}$的所有子集和的集合, 其中 $P_0=\{0\}$。

利用数学归纳法，易证算法 ApproxSubsetSum 第 5 步的修剪 L_i 以及第 6 步删除 L_i 中大于 t 的元素之后，$L_i \subseteq P_i$ 对于 $i=1,2,..,n$ 均成立。于是，第 7 步输出的 z 是 S 的某个子集的和。故算法输出子集和问题的一个近似解。

引理 10.7. 算法 ApproxSubsetSum 输出子集和问题的一个近似解。 □

为分析算法的近似比，先用归纳法证明算法执行过程具有引理 10.8 中的性质。

引理 10.8. $\forall y\in P_i, y\le t$, 存在一个 $y'\in L_i$ 使得$(1-\varepsilon/n)^i y\le y'\le y$。

证明. $i=0$ 时，$L_i=P_i=\{0\}$,引理显然成立。

假设引理在 $i\le k$ 时成立，往证引理在 $i=k+1$ 时成立。由于 $P_{k+1}= P_k \cup P_k+x_{k+1}$,下面区分 $y\in P_k \cap P_{k+1}$ 和 $y\in P_{k+1}- P_k$ 两种情况，分别证明引理成立。

情形 1：$y\in P_k \cap P_{k+1}$, $y\le t$。由于 $y\in P_k$，由归纳假设，存在 $y'\in L_k$ 使得$(1-\varepsilon/n)^k y\le y'\le y$。显然，算法第 4 步产生 L_{k+1} 时，$y'\in L_{k+1}$。由 Trim 算法可知，经算法第 5 步修剪后，存在 $y''\in L_{k+1}$ 使得$(1-\varepsilon/n)y'\le y''\le y'$；因此，$(1-\varepsilon/n)^{k+1}y\le(1-\varepsilon/n)y'\le y''\le y'\le y$,即$(1-\varepsilon/n)^{k+1}y\le y''\le y$，其中 $y''\in L_{k+1}$，引理成立。

情形 2：$y\in P_{k+1}- P_k$，$y\le t$。由 P_{k+1} 的定义知，$y=y'+x_{k+1}$，其中 $y'\in P_k$ 且 $y'\le t$。由归纳假设知，存在 $y''\in L_k$ 使得$(1-\varepsilon/n)^k y'\le y''\le y'$。 于是，

$$(1-\varepsilon/n)^k y'+x_{k+1} \le y''+x_{k+1} \le y \qquad (1)$$

由 $y''\in L_k$ 可知，算法第 4 步产生 L_{k+1} 时，$y''+x_{k+1}\in L_{k+1}$。由 Trim 算法可知，经算法第 5 步修剪后，存在 $y'''\in L_{k+1}$ 使得$(1-\varepsilon/n)(y''+x_{k+1})\le y'''\le(y''+x_{k+1})$, 联立(1)式可知，

$$(1-\varepsilon/n)^{k+1}y' + (1-\varepsilon/n)\, x_{k+1}\le y'''\le y$$

最后，由$(1-\varepsilon/n)^{k+1}(y' + x_{k+1}) \le(1-\varepsilon/n)^{k+1}y' + (1-\varepsilon/n)\, x_{k+1}$ 可知，

$$(1-\varepsilon/n)^{k+1}y\le y'''\le y$$

其中 $y'''\in L_{k+1}$。 □

引理 10.9. 算法 ApproxSubsetSum 是一个 $1+\varepsilon$-近似算法。

证明. 只需证明算法输出的近似解 z 与问题优化解 z^*之间的相对误差不超过ε。

由于 $z^* \in P_n$ 且 $z^*\le t$, 由引理 10.8 可知，存在 $z'\in L_n$ 使得$(1-\varepsilon/n)^n z^*\le z'\le z^*$。再注意到，$z$ 是 L_n 中的最大元素，即 $z'\le z \le z^*$。于是，$(1-\varepsilon/n)^n z^*\le z\le z^*$。

由$(1-\varepsilon/n)^n$ 是 n 的单调递增函数，故$(1-\varepsilon) \le(1-\varepsilon/n)^n$。因此，$(1-\varepsilon)z^*\le z\le z^*$。 □

引理 10.10. 算法 ApproxSubsetSum 的时间复杂度为 $O(n^2(\ln t-\ln z_0)/\varepsilon)$，其中 $z_0=min_{x\in S} x$。

证明. 先推导$|L_i|$的上界。不难发现，$0\in L_i$ 对任意 $i=0,1,2,...,n$ 成立。设经过算法第 5 步和第 6 步操作之后，L_i 具有 $k+2$ 个元素，并记 $L_i=<0,z_1,...,z_{k+1}>$。由于 L_i 经过第 5 步的修剪，由 Trim 算法可知，$z_{j+1}\ge(1-\varepsilon/n)^{-1} z_j$ 对 $j=1,2,...,k$ 均成立。故，$z_{k+1}\ge (1-\varepsilon/n)^{-k} z_1$。又由于 L_i 经过第 6 的操作，故有 $t\ge z_{k+1}\ge (1-\varepsilon/n)^{-k} z_1$。而且，由于 z_1 是 S 中若干个元素之和，因此 $z_1\ge z_0$。所以，$t\ge (1-\varepsilon/n)^{-k} z_0$。由命题 2.16 可解得

$$k \le \frac{\ln t - \ln z_0}{-\ln(1-\varepsilon/n)} \le \frac{n(\ln t - \ln z_0)}{\varepsilon}$$

因此，$|L_i| = k+2 \le 2+ n(\ln t - \ln z_0)/\varepsilon$对任意 $i=0,1,2,...,n$ 成立。

由于算法第 3 步的循环恰好执行 n 次，第 4-6 步的循环体的开销均为 $O(|L_{i-1}|)$。故 ApproxSubsetSum 算法的时间复杂度为 $O(n^2(\ln t - \ln z_0)/\varepsilon)$。 □

由引理 10.7-10.10 得到如下定理。

定理 10.11. 算法 ApproxSubsetSum 是子集和问题的一个完全多项式时间近似模式。□

10.3 基于贪心思想的近似算法

求解优化问题往往涉及多个操作步骤，每个步骤可能由多种策略，贪心策略在每个操作步骤从选择当前最好的策略。这可能会获得优化解（第 5 章），也可能只能获得问题的一个近似可行解。本节通过两个例子介绍贪心策略在近似算法设计中的使用。

10.3.1 集合覆盖问题的近似算法

集合覆盖问题的输入是 n-元集合 X 的一个子集族 $\boldsymbol{F}=\{S_1,S_2,...,S_m\}$，其中 $X=\cup_{S\in\boldsymbol{F}}S$，问题的输出是最小子集族 $\boldsymbol{C}\subseteq F$ 使得 $X=\cup_{S\in\boldsymbol{C}}S$。

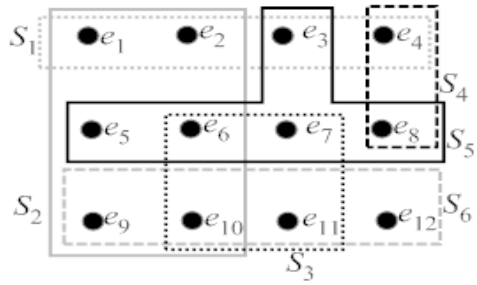


图 10.4 集合覆盖问题实例示意

例如，图 10.4 直观地给出了集合覆盖问题的一个实例，其中 $X=\{e_1,e_2,...,e_{12}\}$，$\boldsymbol{F}=\{S_1,S_2,S_3,S_4,S_5,S_6\}$。不难看到，该实例的最优解为 $C^*=\{S_1,S_5,S_6\}$。利用贪心策略可以快速求得它的一个近似解 $\boldsymbol{C}$。首先，将覆盖元素最多的集合 S_2 添加到 $\boldsymbol{C}$ 中；然后，将覆盖 S_2 之外的点达到最多的集合 S_5 添加到 $\boldsymbol{C}$ 中；再将覆盖 $S_2\cup S_5$ 之外的点达到最多的集合 S_6 添加到 $\boldsymbol{C}$ 中;最后，将覆盖 $S_2\cup S_5\cup S_6$ 之外的点达到最多的集合 S_4 添加到 $\boldsymbol{C}$ 中。此时，X 的 12 个元素均已被覆盖，得到近似解 $C=\{S_2,S_4,S_5,S_6\}$。由这种贪心思想得到如下的近似算法。

集合覆盖的贪心近似算法 GreedySetCover

```
1.U←X，C←∅;                    /* U 表示 X 中尚未被覆盖的元素的集合*/
2.While U≠∅  Do
3.    选取 S∈F 使得|S∩U|最大;   /*贪心地选择覆盖 U 中最多元素的子集 S */
4.    U←U-S;                   /* 将 S 中的所有元素标记为已被覆盖*/
5.    C←C∪{S};                 /* 构造 X 的覆盖 */
6.输出 C
```

算法 GreedySetCover 的时间复杂度为 $O(mn\cdot min(m,n))$，其中$|X|=n$，$|\boldsymbol{F}|=m$。事实

上，算法第 2 步至多循环 $min(m,n)$次；这是因为，每次循环至少覆盖 X 中的一个元素，同时 $\boldsymbol{F}$ 中的一个集合被添加到 $\boldsymbol{C}$ 中。每次循环执行时，第 3 步的开销为 $O(mn)$，因为计算$|S\cap U|$的开销为 $O(n)$且至多 m 个集合可供选择；第 4 步的开销为 $O(n)$，第 5 步的开销为 $O(1)$。

下面，分析算法的近似比。用 $\boldsymbol{C}^*$和 $\boldsymbol{C}$ 分别表示算法的优化解和 GreedySetCover 算法输出的近似解，并根据算法选择的先后次序将 $\boldsymbol{C}$ 中的集合依次记为 $S_1,S_2,...,S_{|\boldsymbol{C}|}$，同时记 $\boldsymbol{C}^*=\{S_1^*,S_2^*,...,S_{|\boldsymbol{C}^*|}^*\}$。分析的目标是利用 $X=\cup_{S\in\boldsymbol{C}}S=\cup_{S\in\boldsymbol{C}^*}S$ 建立$|\boldsymbol{C}^*|$和$|\boldsymbol{C}|$之间的联系。注意到，每个 S_i（$1\le i\le|\boldsymbol{C}|$）被算法选中后，$|\boldsymbol{C}|$增大 1，将代价增量 1 平摊到被 S_i 首次覆盖的所有元素上，每个元素 e 分得代价

$$c_e=1/|S_i-(S_1\cup S_2\cup...\cup S_{i-1})|$$

由此得到 $\sum_{e\in S_i-(S_1\cup S_2\cup...\cup S_{i-1})}c_e=1$。于是，

$$\begin{aligned}|\boldsymbol{C}| &= \underbrace{1+1+...+1}_{|\boldsymbol{C}|个}\\ &= \sum_{e\in S_1}c_e+\sum_{e\in S_2-S_1}c_e+...+\sum_{e\in S_i-(S_1\cup S_2\cup...\cup S_{i-1})}c_e+...+\sum_{e\in S_{|\boldsymbol{C}|}-(S_1\cup S_2\cup...\cup S_{|\boldsymbol{C}|-1})}c_e\\ &= \sum_{e\in X}c_e\\ &\le \sum_{e\in S_1^*}c_e+\sum_{e\in S_2^*}c_e+...+\sum_{e\in S_{|\boldsymbol{C}^*|}^*}c_e \qquad (\text{因为 } e\in X \text{ 可能被 } \boldsymbol{C}^*\text{的多个集合覆盖})\\ &\le |\boldsymbol{C}^*|\cdot f(n) \qquad (\text{只要}\sum_{e\in S_i^*}c_e\le f(n)\text{对 } 1\le i\le|\boldsymbol{C}^*|\text{成立})\end{aligned}$$

由此可见，只要$\sum_{e\in S_i^*}c_e\le f(n)$对 $1\le i\le|\boldsymbol{C}^*|$成立，则 $f(n)$即是近似比。下面的引理给出了更强的结论。

引理 10.12. 对于任意 $S\in\boldsymbol{F}$ 有 $\sum_{e\in S}c_e\le H(|S|)\le H(n)$, 其中 $H(n)=\sum_{j=1}^{n}\frac{1}{j}$。

证明. 为方便计，令 $u_0=|S|$，$u_i=|S-(S_1\cup S_2\cup...\cup S_i)|$，其中 $1\le i\le|\boldsymbol{C}|$。事实上，$u_i$表示算法选取集合 $S_1,S_2,...,S_i$之后，S 中仍未被覆盖的元素的个数。于是，$u_0,u_1,...,u_{|\boldsymbol{C}|}$是单调递减整数数列且 $u_{|\boldsymbol{C}|}=0$,令 $k=min_{u_i=0}\,i$ 是该序列中首个 0 元素出现的位置。这意味着，算法选取 S_k之后，S 中所有元素均被覆盖；在这之前算法选择 S_i（却未选取 S）添加到 $\boldsymbol{C}$ 意味着 $u_{i-1}=|S-(S_1\cup S_2\cup...\cup S_{i-1})|<|S_i-(S_1\cup S_2\cup...\cup S_{i-1})|$, $1\le i\le k$。因此

$$\begin{aligned}\sum_{e\in S}c_e &= \sum_{e\in S\cap S_1}c_e+\sum_{e\in S\cap(S_2-S_1)}c_e+...+\sum_{e\in S\cap(S_k-(S_1\cup...\cup S_{k-1}))}c_e\\ &=\sum_{e\in S\cap S_1}\frac{1}{|S_1|}+\sum_{e\in S\cap S_1}\frac{1}{|S_2-S_1|}+...+\sum_{e\in S\cap(S_k-(S_1\cup...\cup S_{k-1}))}\frac{1}{|S_k-(S_1\cup...\cup S_{k-1})|}\\ &\le\sum_{e\in S\cap S_1}\frac{1}{|S|}+\sum_{e\in S\cap S_1}\frac{1}{|S-S_1|}+...+\sum_{e\in S\cap(S_k-(S_1\cup...\cup S_{k-1}))}\frac{1}{|S-(S_1\cup...\cup S_{k-1})|}\\ &=\frac{u_0-u_1}{u_0}+\frac{u_1-u_2}{u_1}+...+\frac{u_{k-1}-u_k}{u_{k-1}}\\ &\le[H(u_0)-H(u_1)]+[H(u_1)-H(u_2)]+...+[H(u_{k-1})-H(0)]\\ &=H(u_0)\end{aligned}$$

□

根据前面的分析和第 2 章例 2.9 中函数 $H(n)$的性质，得到如下定理。

定理 10.13. 算法 GreedySetCover 的近似比为 $\ln n+1$。 □

10.3.2 不相交路径问题的近似算法

不相交路径问题的输入是图 $G=(V,E)$和一个源-汇顶点对集合 $A\subseteq V\times V$，其中$|E|=m$，输出是 A 的一个最大子集 B 使得 B 中的所有顶点对在 G 中存在无公共边的路径。

如下的贪心策略可以给出不相交路径问题的一个近似解。选择$(u,v)\in A$ 使得图 $G=(V,E)$中 u,v-最短路径 $P_{u,v}$ 的长度达到最小值，将(u,v)添加到输出集合 B。此后，将路径 $P_{u,v}$ 上的边从图 G 中删除，因为不能再用这些边连接 A 中的其他顶点对。选择边数最少的路径 $P_{u,v}$ 可以让尽可能多的边保留在图中，用于连接更多的其他顶点对。上述思想可以形式化为下面的算法。

不相交路径近似算法 EdgeDisjointPath

输入： 图 $G=(V,E)$和一个源-汇顶点对集合 $A\subseteq V\times V$

输出： $B\subseteq A$ 使得$|B|$最大且 B 中的所有顶点对在 G 中存在无公共边的路径

1. $B\leftarrow\varnothing$;
2. Do
3. 　　在图 G 上为$\forall(s,t)\in A$ 计算最短路径 $P_{s,t}$，得到集合 $\boldsymbol{P}=\{P_{s,t}|(s,t)\in A\}$;
4. 　　从 $\boldsymbol{P}$ 中选择长度最短(即边数最少)的路径 $P_{u,v}$;
5. 　　$B\leftarrow B\cup\{(u,v)\}$; 　$A\leftarrow A-\{(u,v)\}$;
6. 　　从 G 上删除 $P_{u,v}$ 中所有的边;
7. Until $\boldsymbol{P}=\varnothing$
8. 输出 B

不相交路径算法的时间复杂度为 $O(|A||V|^4)$。事实上，算法第 2-7 步的循环至多执行$|A|$遍，因为循环每执行一遍 A 中顶点对至少减少 1（只要算法未结束）。在循环体中，第 3 步可通过调用 all-pairs 最短路径算法实现，其开销为 $O(|V|^4)$；第 4-6 步的开销为 $O(|E|)$。

下面分析算法的近似比。用 B^*和 B 分别表示优化解和算法输出的近似解。分析过程中，有时发生符号混用。例如，$B^*\subseteq A$ 是源-汇顶点对构成的集合，但有时也用 B^*表示这些源-汇顶点对之间互不相交的路径构成的集合；B 也有类似的用法。

分析过程的思路如下。通过参数 k 将 B^*划分为两个部分，将每个部分的大小用 $|B|$和 k 表示出来，从而建立表达式$|B^*|\le g(k)|B|$, 最后通过选取参数 k 使得 $g(k)$取值最小来得到算法的近似比 $min_k\, g(k)$。

给定参数 k。对于 B^*中的任意路径，如果其长度（即边的条数）大于 k，则称之为长路径，否则称之为短路径。

先考察 B^*中长路径的条数。由于 B^*是问题的优化解，所以任意两条路径均无公共边。而且，含于 B^*中所有长路径中边至多为$|E|=m$ 条，每条长路径的边数均大于 k,故长路径至多有 m/k 条。由$|B|\ge 1$ 得到如下引理。

引理 10.14. 对于任意 k，B^*中长路径至多有 $\dfrac{m}{k}|B|$条，其中$|E|=m$。 □

再考察 B^*中短路径的条数。设 $P_{opt}\in B^*$是一条任意的短路径，则 P_{opt} 必然与 B 中的至少一条路径相交（具有公共边）；若不然，EdgeDisjointPath 算法结束后，P_{opt}

所有边仍可用于连接 A 中顶点对，这与算法终止条件矛盾。

利用上述结论，容易建立 B^*中短路径的条数与$|B|$之间的关系。由于 P_{opt} 是短路径，它至多有 k 条边。前面的分析表明，P_{opt} 与 B 中的至少一条路径相交（具有公共边）。设 P 是 EdgeDisjointPath 算法选中的首条与 P_{opt} 相交的路径。算法选中 P 而未选择 P_{opt}，说明 P 的长度小于 P_{opt} 的长度。因此，P 中至多含有 k 条边。同时，由于 B^*中的短路径无公共边，故 B^*中与 P 相交的路径至多有 k 条。于是，

短路径条数$\le k \cdot |\{P|P\in B$ 是算法选中的首条与 P_{opt} 相交的路径, $P_{opt}\in B^*$是短路径$\}|$
$\le k \cdot |B|$

引理 10.15. 对于任意 k，B^*中短路径至多有 $k|B|$条。 □

由引理 10.14 和引理 10.15 可得$|B^*| \le \left(\frac{m}{k}+k\right)\cdot|B|$。注意，$\frac{m}{k}+k$ 在 $k=\sqrt{m}$ 取最小值 $2\sqrt{m}$。故得到算法 EdgeDisjointPath 的近似比为 $2\sqrt{m}$。

定理 10.16. 算法 EdgeDisjointPath 的近似比为 $2\sqrt{m}$。 □

10.4 基于局部搜索的近似算法

局部搜索被广泛用于近似求解难解的优化问题。其基本思想是，从任意（或精心选取的）一个可行解出发，不断对其进行局部的细微改动使得优化问题的目标函数取值逐步达到局部最优，最后将获得的局部最优解作为近似解输出。

优化问题的解空间可以表示为一个图，其中每个顶点表示问题的一个可行解，两个可行解之间存在一条边当且仅当这两个可行解经过某种局部的细微改动可以相互转换。如果一个可行解的代价大于等于其所有“相邻”可行解的代价，则该可行解是一个局部最优解。如果一个可行解的代价优于任意其他可行解的代价，则该可行解是一个全局最优解。

在利用局部搜索技术设计近似解时，要求问题解空间对应的图上顶点的度较小（比如，具有多项式界限）。因此，在每个可行解上仅需多项式步局部修改即可获得代价更优的“相邻”可行解（如果存在的话）。

为了证明局部搜索近似算法能够在多项式时间内终止，必须证明算法从任意可行解出发至多经过多项式步操作即可获得一个局部最优解。仅证明图的直径是多项式有界的不足以证明局部搜索算法能够在多项式时间内终止。

10.4.1 最大割问题的近似算法

虽然最小割问题可以在多项式时间内求解（参见第 7.1 节），最大割问题却是 *NP*-完全问题。本小节利用局部搜索技术给出最大割问题的一个 2-近似算法。

最大割问题的输入是一个加权图 $G=(V,E)$,其中每条边$(u,v)\in E$ 具有正整数权值 w_{uv},输出顶点集的一个子集 S 使得 $\sum_{u\in S, v\in V-S} w_{uv}$ 达到最大值。

直接应用局部搜索策略可得到最大割问题的一个近似算法。首先，将割初始化为 $S=\{u\}$，其中 u 是 V 中的任意顶点，显然介于 S 和 V-S 间的所有边是一个割，故 S 是问题的一个可行解。接下来，利用如下局部修改获得一个局部优化解作为问题的近似解：从 V 中选择一个顶点 v,交换 v 在 S 和 V-S 中的位置。交换的含义是，如果 $v\in S$ 则令 $S=S-\{v\}$；否则，令 $S=S\cup\{v\}$。当然，要求这种交换能够使得目标函数

的取值增大，这只需检查如下定义的交换代价 *cost*(*v*,*S*)是否大于 0。*cost*(*v*,*S*)定义了交换顶点 *v* 时新出现的割边的总代价减去消失的割边的总代价。

$$\cos t(v,S)=\begin{cases}\sum_{u\in S}w_{uv}-\sum_{u\in V-S}w_{uv} & v\in S\\ \sum_{u\in V-S}w_{uv}-\sum_{u\in S}w_{uv} & v\in V-S\end{cases}$$

最大割近似算法 ApproxMaxCut

输入:加权图 $G=(V,E)$,其中每条边$(u,v)\in E$ 具有正整数权值 w_{uv}

输出：顶点集的一个子集 S 使得$\sum_{u\in S,v\in V-S}w_{uv}$ 达到最大值

1.$S\leftarrow\{u\}$ /*u 是 V 中任意顶点*/

2.repeat

3. 任取一个满足 *cost*(*v*,*S*)>0 的顶点 $v\in V$, 交换 v 在 S 和 V-S 中的位置；

4. until 不存在 $v\in V$ 使得 *cost*(*v*,*S*)>0

5.输出 S

首先，分析算法的时间复杂度。算法第 3 步在最坏情况下需要为任意 $v\in V$ 计算代价增量 *cost*(*v*,*S*), 而计算 *cost*(*v*,*S*)的最坏开销为|*V*|,故每次执行第 3 步的开销为 O(|*V*|^2)。由于第 3 步每执行一遍，可行解的代价至少增大 1（因为所有边的权值均是正整数），并且可行解的代价以$\sum_{uv\in E}w_{uv}$ 为上界，故第 2-4 步至多循环$\sum_{uv\in E}w_{uv}$ 遍。于是，算法的时间复杂度为 O(|*V*|$^2\sum_{uv\in E}w_{uv}$)。例如，当所有边的权值为 1 时，算法的运行时间为 O(|*V*|^4)。

下面分析算法的近似比，用 S^*和 S 分别表示优化解和算法给出的近似解。不难看到，$w(S^*)\le\sum_{uv\in E}w_{uv}$。

另一方面，由于 S 是局部最优解，故$\forall v\in S$ 必有 *cost*(*v*,*S*)<0,即

$$\sum_{u\in S}w_{uv}<\sum_{u\in V-S}w_{uv}$$

于是，

$$\sum_{u\in V-S}w_{uv}+\sum_{u\in S}w_{uv}<2\cdot\sum_{u\in V-S}w_{uv}$$

因此，

$$\frac{1}{2}\cdot\sum_{u\in V}w_{uv}<\sum_{u\in V-S}w_{uv}\qquad\forall v\in S\qquad(1)$$

同理可得，$\forall v\in$V-S 必有

$$\frac{1}{2}\cdot\sum_{u\in V}w_{uv}<\sum_{u\in S}w_{uv}\qquad\forall v\in\text{V-}S\qquad(2)$$

联立(1)式和(2)式得到，

$$\sum_{uv\in E}w_{uv}<2\cdot w(S)$$

最终得到 $w(S^*)\le\sum_{uv\in E}w_{uv}<2\cdot w(S)$。

定理 10.17. 算法 Approx-Max-Cut 的近似比为 2。 □

10.4.2 设施定位问题的近似算法

直观上，设施定位问题考虑如何代价最优地开启设施以便向指定用户提供服

务。具体地讲，开启部分设施并将每个用户指派到距离最近的设施接受服务。开启每个设施具有一定的代价，每个用户与向其服务的设施之间具有路由代价，问题求解的目标是有选择地开启部分设施向所有用户提供服务使得总代价（包括开启设施的代价和路由代价）最小。问题的形式化定义如下。计算路由代价需要考虑用户和设施之间的距离，本节讨论的设施定位问题要求距离函数满足非负性、对称性和三角不等式。

设施定位问题

输入：设施集合 F,用户集合 U,距离函数 d:$F\times U\to R^+$和开启设施 i 的代价 f_i, i=1,...,$|F|$。给定 $S\subseteq F$, 定义开启 S 中所有设施的代价为 $C_f(S)=\sum_{i\in S} f_i$。定义 S 向 $j\in U$ 提供服务的代价为 $r_j^S=min_{\,i\in S}\, d(i,j)$,定义 S 向 U 服务的代价为 $C_r(S)=\sum_{j\in U}\, r_j^S$。定义开启 S 向 U 服务的总代价为 $C(S)=C_f(S)+C_r(S)$。

输出：$S\subseteq F$ 使得 $C(S)$达到最小值。

直接应用局部搜索策略可得到设施定位问题的一个近似算法。首先，将 S 初始化为 F 的任意子集，它也是问题的一个可行解。接下来，利用局部修改获得逐步获得一个局部优化解作为问题的近似解。允许的局部修改操作包括：（1）插入操作，亦即将 F-S 中的一个设施加入 S 中；（2）删除操作，亦即从 S 中删除一个设施；（3）替换操作，亦即将 S 中的一个设施替换为 F-S 中的一个设施。同 10.4.1 节类似，局部操作必须确保目标函数的取值减小。算法的形式化描述如下。

设施定位问题的近似算法 LocalSearchFL

输入：设施集 F,用户集 U，距离函数 d:$F\times U\to R^+$和开启设施 i 的代价 f_i, i=1,...,$|F|$

输出：$S\subseteq F$ 使得 $C(S)$达到最小值

1. 将 S 初始化为 F 的任意子集；
2. 若存在插入（或替换、删除）操作使得 S 的代价下降因子$1-\frac{\varepsilon}{p(n)}$,则执行该操作；

   /*$p(n)$是 $n=|F|+|U|$的多项式（如 $p(n)=n^2$）用于控制算法运行时间和近似比*/
3. 重复第 2 步直到任意插入、替换、删除操作均不满足条件，输出 S

首先断言算法 LocalSearchFL 必然在有限步骤内停止。由设施定位问题的定义可知，问题的任意可行解的代价不超过 $q_1(n,f,d)=\sum_{i\in F}\, f_i+n\cdot max\{\, r_j^F \mid j\in U\}$，其中 $n=|F|+|U|$;而且，问题的任意可行解的代价至少为 $q_2(n,f,d)=min_{i\in F}\, f_i+\sum_{j\in U} r_j^F$。令 $q(n,f,d)=q_1(n,f,d)/q_2(n,f,d)$。由于第 2 步每执行一遍，可行解 S 的代价至少下降因子 $(1-\varepsilon/p(n))$,因此算法第 2 步运行 $k\cdot p(n)/\varepsilon$ 遍之后，可行解代价 $C(S)$满足

$$q_2(n,f,d)\le C(S)\le \left(1-\frac{\varepsilon}{p(n)}\right)^{\frac{p(n)}{\varepsilon}k}\cdot q_1(n,f,d)\le \frac{q_1(n,f,d)}{e^k}$$

上述不等式用到结论“$(1-x^{-1})^x\le e^{-1}$ 对$\forall x>1$ 成立”。于是，$k\le \ln q(n,f,d)$。亦即，算法第 2 步在运行至多 $\frac{p(n)}{\varepsilon}\cdot \ln q(n,f,d)$遍之后，算法必然终止。

至此得到，算法运行在 n 的多项式时间内终止，因为算法第 2 步每运行一遍所需的时间也是 n 的多项式（细节留给读者）。

下面分析算法的近似比。引理 10.18 和引理 10.19 分别给出了任意局部优化解的路由总代价 $C_r(S)$和开启总代价 $C_f(S)$的上界。利用引理 10.18 和引理 10.19，关联算法操作的特性，最后证明算法的近似比为 5（定理 10.20）。为证明引理 10.18 和引理 10.19，在局部优化解 S 和全局优化解 S^*上，引入下列记号。

$\forall j\in U, \sigma^*(j)\in S^*$表示 S^*中向用户 j 提供服务的设施

$\forall j\in U, \sigma(j)\in S$ 表示 S 中向用户 j 提供服务的设施

$\forall j\in U, r^*(j)$表示用户 j 到设施 $\sigma^*(j)$的距离，即 $r^*(j)=d(\sigma^*(j),j)=min_{i^*\in S^*}d(i^*,j)$

$\forall j\in U, r(j)$表示用户 j 到设施 $\sigma(j)$的距离，即 $r(j)=d(\sigma(j),j)=min_{i\in S}d(i,j)$

引理 10.18.设 S 和 S^*分别是局部优化解和全局优化解，则 $C_r(S)\le C(S^*)$。

证明. $\forall i^*\in S^*-S$,由于 S 是局部优化解,将 i^*插入 S 不会导致代价下降,故

$$f_{i^*}+\sum_{j:\sigma^*(j)=i^*}(r^*(j)-r(j))\ge 0$$

$\forall i^*\in S^*\cap S$,则必有

$$f_{i^*}+\sum_{j:\sigma^*(j)=i^*}(r^*(j)-r(j))=0$$

将上述两个表达式在$\forall i^*\in S^*$上做和得到

$$C_f(S^*)+C_r(S^*)-C_r(S)\ge 0$$

亦即，$C_r(S)\le C_f(S^*)+C_r(S^*)=C(S^*)$。 □

引理 10.19.设 S 和 S^*是局部优化解和全局优化解，则 $C_f(S)\le 2C(S^*)+2C_r(S)\le 4C(S^*)$。

证明.为了给出 $C_f(S)$的上界，考虑将 S 中的设施 i 替换为 S^*中距离 i 最近的设施 i^*，并将替换前由 i 提供服务的用户全部重新分配给 i^*。由于 S 是局部优化解，故上述操作不会导致代价下降；这意味着，设施 i 的开启代价 f_i的一个上界是路由代价的增量加上设施 i^*的开启代价，即

$$f_{i^*}-f_i+\sum_{j:\sigma(j)=i}(d(j,i^*)-d(j,i))\ge 0 \qquad \forall i\in S$$

然而，由于同一 i^*可能与 S 中的不同设施均距离最近，故同一 i^*可能出现在不同 i 对应的不等式中。为避免这一情况，让$\forall i^*\in S^*$仅对应于 S 中的一个设施。为此，令 $N(i^*)=\{i \mid i\in S$ 且 $d(i,i^*)=min_{k\in S^*}d(i,k)\}$，它由 S 中所有可能与 i^*进行替换操作的设施构成。称 $N(i^*)$中距离 i^*最近的设施为 i^*的**主设施**，记为 $\pi(i^*)$，亦即 $d(i^*,\pi(i^*))=min_{i\in N(i^*)}d(i^*,i)$。 $N(i^*)$中其余设施称为 i^*的**次设施**。

为清晰记，对于$\forall i\in S$，用 R_i 表示 i 向用户提供服务的总路由代价，即 $R_i=\sum_{j:\sigma(j)=i}r_j$，其中 $r_j=d(i,j)$。替换操作后满足 $\sigma(j)=i$ 的用户将由 i^*提供服务，故 i^*向用户提供服务的总路由代价为 $R_i^*=\sum_{j:\sigma(j)=i}r_j^*$，其中 $r_j^*=d(i^*,j)$。需要特别强调的是，R_i 和 R_i^*求和的下标集是一样的。下面是关于主设施和次设施的两个论断。

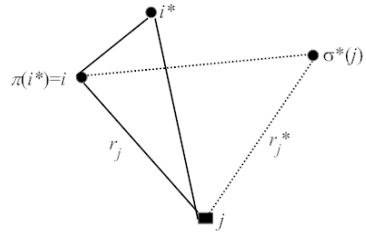
论断 1.如果 i 是 i^*的主设施，则 $f_i\le R_i+R_i^*+f_{i^*}$。

论断 2.如果 i 是 i^*的次设施，则 $f_i\le 2(R_i+R_i^*)$。

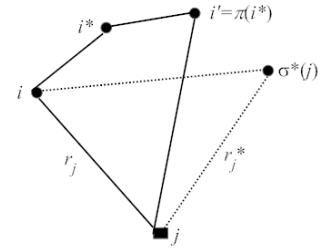
由论断 1 和论断 2 可知，引理成立，因为

$$C_f(S) \le C_f(S^*)+2\sum_{i\in S}(R_i+R_i^*)$$
$$= C_f(S^*)+2C_r(S^*)+2C_r(S)$$
$$\le 2C(S^*)+2C_r(S)$$
$$\le 4C(S^*)$$

上述分析表明，为证明引理，仅需分别证明论断 1 和论断 2 成立。



(a)论断 1 证明涉及的数据对象



(b)论断 2 证明涉及的数据对象

图 10.5 证明论断 1 和论断 2 涉及的数据对象示意图

论断 1 的证明。考虑将 S 中的设施 i 替换为 i^*,并将由 i 提供服务的每个用户 j（满足 $\sigma(j)=i$）均改由 i^*向其提供服务。由于 S 是局部优化解，上述操作不会引起 S 的代价下降，故有

$$f_{i^*} - f_i + \sum_{j:\sigma(j)=i}(d(j,i^*)-d(j,i)) \ge 0$$

注意，上述不等式左端的每个求和项满足

$$d(j,i^*)-d(j,i) \le d(i^*,i) \quad \text{(三角不等式)}$$
$$\le d(i,\sigma^*(j)) \quad (S^*\text{中距离 } i \text{ 最近的设施为 } i^*)$$
$$\le (r_j+r_j^*) \quad \text{(三角不等式)}$$

将上述不等式带入，再由 R_i 和 R_i^*的定义即得，$f_{i^*}-f_i+R_i+R_i^*\ge 0$；亦即论断 1 成立。

论断 2 的证明。设 $i'=\pi(i^*)$，即 i'是 i^*的主设施。由主设施的定义知，$d(i^*,i')\le d(i^*,i)$。考虑从 S 中删除设施 i，并将其服务的每个用户 j 改由 i'服务，于是用户 j 的路由代价的增量为

$$d(i',j)-d(i,j) \le d(i',i) \quad \text{(三角不等式)}$$
$$\le d(i,i^*)+d(i^*,i') \quad \text{(三角不等式)}$$
$$\le 2d(i,i^*) \quad (d(i^*,i')\le d(i^*,i))$$
$$\le 2d(i,\sigma^*(j)) \quad (S^*\text{中距离 } i \text{ 最近的设施为 } i^*)$$
$$\le 2(r_j+r_j^*) \quad \text{(三角不等式)}$$

由于 S 是局部优化解，上述操作不会使得 S 代价下降,因此

$$-f_i + \sum_{j:\sigma(j)=i}(d(i',j)-d(i,j)) \ge 0$$

亦即，$-f_i+2(R_i+R_i^*)\ge 0$。由此可知，论断 2 成立。 □

定理 10.20. 算法 LocalSearchFL 的近似比为 $5+O(\varepsilon)$。

证明. 算法第 2 步被执行时，算法从一个可行解得到另一个可行解，代价至少下降因子$(1-\varepsilon/p(n))$。因此，算法未必终止于一个局部优化解。然而，修改引理 10.18 和引理 10.19 的证明，仍然可以给出算法输出的解 S 的代价界限。事实上，算法终止时，由于任意操作均不能使得 S 的代价下降超过 $C(S)\cdot\varepsilon/p(n)$,因此将引理 10.18 和引

理 10.19 证明过程中所有不等式右端的 0 全部替换成-$C(S)\cdot\varepsilon/p(n)$之后仍成立。证明中涉及的不等式不超过 n^2 个，且证明过程均只涉及不等式累加，故重复同样的论述过程可得

$$(1-\varepsilon n^2/p(n))C(S)\le 5C(S^*)$$

取 $p(n)=n^2$ 即得，$C(S)/C(S^*)\le 5/(1-\varepsilon)=5+O(\varepsilon)$。 □

对算法 LocalSearchFL 进行更细致的分析，可以证明其近似比为 $3+O(\varepsilon)$。但过程比本节介绍的过程复杂得多，感兴趣的读者可参考文献[13]。此处，作为教学目的，仅给出了一个较宽松的近似比。此外，设施定位问题目前获得的最佳近似比为 1.52[14]。

10.5 基于动态规划的近似算法

动态规划算法（第 4 章）利用优化问题的优化子结构和重叠子问题性质自底向上地计算优化解的代价，然后由优化解的结构构造出优化解。一般而言，利用动态规划方法求解问题的近似解时有两种策略，它们均不是直接将动态规划方法应用到问题实例上。

第一种策略是将问题实例 I 变形，得到具有特殊性质的问题实例 I'；再将动态规划算法运用到变换后的问题实例 I'上求得 I'的优化解；再利用 I'的优化解得到原来实例 I 的一个近似解。近似解的近似比取决于变形过程的性质。10.5.1 将采用这种技术给出 0-1 背包问题的完全多项式近似模式。

第二种策略将动态规划算法视为枚举问题解空间的一个过程，仅运用动态规划方法枚举问题解空间的一个子空间，将得到问题的一个近似解。此时，近似算法的近似比取决于近似算法搜索的子空间相对于问题的整个解空间的“间隙”的大小。10.5.2 节将采用这种技术给出装箱问题的近似模式。

10.5.1 0-1 背包问题的完全多项式近似模式

给定一个容量为 C 的背包和 n 个物品，其中第 i 个物品的重量为 w_i,其价值为 v_i，0-1 背包问题要求对每个物品选择放入背包和不放入背包使得背包内物品总重量不超过 C 且总价值达到最大值。

在 4.4 节中，我们观察到 0-1 背包问题具有优化子结构和重叠子问题等性质，然后利用规模为 $n\times C$ 的二维数组自底向上地求解 0-1 背包问题的子问题，得到了复杂度为 $\Theta(nC)$的动态规划算法，其中 n 是问题实例中物品的个数，C 是背包容量。注意，该算法是一个伪多项式时间算法，而不是多项式时间算法，因为时间复杂度 $\Theta(nC)$不是输入规模 n 和 $\log C$ 的多项式。0-1 背包问题已被证明是 *NP*-完全问题，它不太可能存在多项式时间的精确算法。

0-1 背包问题的动态规划算法还可以改造成如下形式。令 $b_{i,j}$ 表示将重量分别为 $w_i,w_{i+1},...,w_n$ 价值分别 $v_i,v_{i+1},...,v_n$ 的物品装入背包中欲获得总价值量 j 所需的最小背包容量,其中 $1\le j\le \sum_{i=1}^{n} v_i$ 。容易证明，0-1 背包问题的优化子结构可重新表述如下

$$b_{i,j}=\begin{cases} b_{i+1,j} & j<v_i, i<n \\ \min\{b_{i+1,j}, b_{i+1,j-v_i}+w_i\} & j\ge v_i, i<n \end{cases}$$

$$b_{n,j}=\begin{cases} w_n & j \le v_n \\ +\infty & j > v_n \end{cases}$$

于是，利用规模为 $n\times\sum_{i=1}^{n}v_i$ 的二维数组存储优化解的代价，可以采用类似于 4.4 节的过程自底向上地求解 0-1 背包问题的子问题，并构造优化解。下面，将该算法称为 BoolPacking 算法,其时间复杂度为 $\Theta(n\sum_{i=1}^{n}v_i)$。它算法仍是一个伪多项式时间算法。实现细节留给读者。

现在，考虑 0-1 背包问题的近似求解。近似算法的基本思想是，将问题的每个输入实例 $I=\langle C;w_1,w_2,...,w_n;v_1,v_2,...,v_n\rangle$ 变换成新的实例 $I'=\langle C;w_1,w_2,...,w_n;v_1',v_2',...,v_n'\rangle$，使得 $\sum_{i=1}^{n}v_i'$ 以 n 的一个多项式为上界；然后利用 BoolPacking 算法求解 I'，将它的解作为原问题的近似解输出。具体地讲，给定变形参数 ε,令 $K=n/\varepsilon$, $v_{max}=max_{1\le i\le n}v_i$，对原问题中所有物品的价值 v_i 进行缩放得到 v_i',即 $v_i'=\lfloor v_i\cdot(K/v_{max})\rfloor$。于是，有

$$\sum_{i=1}^{n}v_i'=\sum_{i=1}^{n}\left\lfloor\frac{v_i}{v_{\max}}\cdot K\right\rfloor\le\sum_{i=1}^{n}\frac{v_i}{v_{\max}}\cdot K\le Kn=n^2/\varepsilon$$

因此，BoolPacking 算法求解问题实例 I'的时间复杂度为 $\Theta(n\sum_{i=1}^{n}v_i')=\Theta(n^3/\varepsilon)$。

至此，得到如下的时间复杂度为 $\Theta(n^3/\varepsilon)$的近似算法。

算法 ApproxBoolPacking

输入：正整数重量数组 $W[1:n]$和正整数价值量数组 $V[1:n]$,正整数容量 C,参数 ε

输出：0-1 背包问题的一个 1+ε 近似解

1.$K\leftarrow n/\varepsilon$;

2.$v_{max}\leftarrow max_{1\le i\le n}V[i]$;

3.$V[i]\leftarrow\lfloor V[i]\cdot(K/v_{max})\rfloor$

4.$\langle x_1,x_2,...,x_n\rangle\leftarrow$BoolPacking($C$,$W[1:n]$,$V[1:n]$)； /*调用动态规划算法*/

5.输出 $\langle x_1,x_2,...,x_n\rangle$ 作为原问题的近似解

定理 10.21.算法 ApproxBoolPacking 是一个 $1+\varepsilon$-近似算法。

证明.在问题实例 $I=\langle C;w_1,w_2,...,w_n;v_1,v_2,...,v_n\rangle$ 上，设 0-1 向量 $\langle x_1,x_2,...,x_n\rangle$ 是算法 ApproxBoolPacking 输出的近似解，而 0-1 向量是 $\langle y_1,y_2,...,y_n\rangle$ 是问题实例 I 的最优解。注意，$\langle x_1,x_2,...,x_n\rangle$ 也是问题实例 $I'=\langle C;w_1,w_2,...,w_n;v_1',v_2',...,v_n'\rangle$ 的优化解。

令 $z_{opt}=\sum_{i=1}^{n}y_iv_i$，$z'_{opt}=\sum_{i=1}^{n}y_iv_i'$，$z'=\sum_{i=1}^{n}x_iv_i'$，$z=\sum_{i=1}^{n}x_iv_i$。由于 $v_i'=\lfloor v_i\cdot(K/v_{max})\rfloor$ 对 $1\le i\le n$ 成立，故 $\frac{K}{v_{\max}}z\ge z'$。再由 $\langle x_1,x_2,...,x_n\rangle$ 是问题实例 I'的优化解可知，$z'\ge z'_{opt}$。联立上述两个不等式得到

$$\begin{aligned} z&\ge\frac{v_{\max}}{K}z'_{opt}\\ &=\frac{v_{\max}}{K}\sum_{i=1}^{n}y_iv_i'\\ &=\frac{v_{\max}}{K}\sum_{i=1}^{n}\left\lfloor\frac{K}{v_{\max}}v_i\right\rfloor y_i\\ &\ge\frac{v_{\max}}{K}\sum_{i=1}^{n}\left(\frac{K}{v_{\max}}v_i-1\right)y_i\\ &=\sum_{i=1}^{n}y_iv_i-\varepsilon\cdot v_{\max}\end{aligned}$$

$\geq z_{opt'}(1-\varepsilon)$ □

10.5.2 装箱问题的多项式近似模式

装箱问题(参见 10.2.2 节)要求将体积分别为 $a_1,a_2,\ldots,a_n\in(0,1]$的 n 个物品装入容积为 1 的箱子使得所用箱子个数最少。

为了近似求解装箱问题，对于装箱问题的任意输入实例 $I=\langle a_1,a_2,...,a_n\rangle$ 和相对误差界限 $\varepsilon<1$，先将 I 转换成另外三个实例 I',I^{down} 和 I^{up} 使得 I^{up} 中物品体积的不同取值的个数是固定值 $K=1/\varepsilon^2$。然后，利用动态规划算法在 n 的多项式时间内搜索 I^{up} 的解空间，获得 I^{up} 的最优解。最后，将 I^{up} 的优化解转换为 I 的一个近似解。

装箱问题近似算法 ApproxBinPacking(I,ε)
输入：装箱问题的实例 I 和相对误差参数 $\varepsilon<1$
输出：I 的一个近似最优的装箱方案；
1. $I',I^{down},I^{up}\leftarrow$Transfrom($I,\varepsilon$)；
2. $S\leftarrow$DynamicSearch($I^{up},1/\varepsilon^2$);
3. $S'\leftarrow$SolutionTrans(S,I,ε);
4. 输出 S;

下面，先分别介绍 ApproxBinPacking 算法的第 1-3 步,再给出算法分析。

问题实例的变换

直观上，装箱问题的最优解的代价受体积较小的物品的影响极小，因为多个体积较小的物品可以装入少数几个箱子内。因此，可以从输入实例 I 中删除体积小于 ε 的所有物品，得到 I'。然后，将剩余物品分为 $K=1/\varepsilon^2$ 组，每组中物品的体积大致相当，且每组包含相同个数的物品（最后一组除外）。将每组中所有物品的体积分别用该组内的最大体积和最小体积替换，则得到 I^{up} 和 I^{down}。变换过程形式化地描述为如下的算法 Transform(I,ε)。

装箱问题实例变化算法 Transform(I,ε)
输入：装箱问题的实例 I 和变换参数 $\varepsilon<1$
输出：变形后的三个实例 I',I^{down} 和 I^{up}
1. 删除 I 中所有体积小于 ε 的物品，得到 I'，记 $n=|I'|$;
2. 将 I'中所有物品按体积大小递增排序，划分为 $K=1/\varepsilon^2$ 组，每组 $n\varepsilon^2$ 个物品；
3. 将每组中每个物品的体积修改为该组内最大的物品体积，得到实例 I^{up};
4. 将每组中每个物品的体积修改为该组内最小的物品体积，得到实例 I^{down};
5. 输出 I',I^{down} 和 I^{up};

显然，算法 Transfrom(I,ε)的时间复杂度为 $O(n\log n)$。并且，变换后的问题实例具有引理 10.22 所述的性质，其中 $Opt(I')$,$Opt(I^{up})$和 $Opt(I^{down})$分别表示 I',I^{down} 和 I^{up} 的优化解的代价。

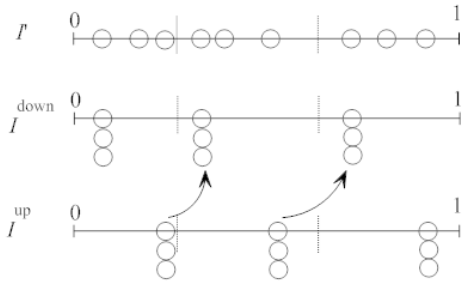
引理 10.22.算法 Transfrom(I,ε)输出的实例 I',I^{down} 和 I^{up} 满足如下性质。
(1).每个箱子至多容纳 I',I^{down} 和 I^{up} 的 $L=\lceil 1/\varepsilon\rceil$ 个物品；
(2).I^{down} 和 I^{up} 中物品体积至多有 $K=1/\varepsilon^2$ 个不同的取值；
(3).$Opt(I^{down})\leq Opt(I')$，且 $n\varepsilon\leq Opt(I')$,其中 n 是 I'中的物品个数；

(4).$Opt(I^{up})\le(1+\varepsilon)Opt(I')$；

证明.(1)和(2)由变换算法第 1 步和第 2 步操作过程即得。

由于实例 I'的每个可行解也是实例 I^{down} 的一个可行解，故 $Opt(I^{down})\le Opt(I')$。由于 I'中每个物品的体积至少为 ε，故其中物品的总体积为至少为 $n\varepsilon$,故 $n\varepsilon\le Opt(I')$。因此，(3)成立。

往证(4)。首先，注意到 I^{up} 中第 i 组的每个物品的体积不超过 I^{down} 中第 i+1 组的每个物品的体积（如下图所示）。这意味着，将实例 I^{down} 的优化解 S 进行修改可



以获得 I^{up} 的一个可行解 S'：先从 S 使用的各个箱子中删除 I^{down} 中第一组物品；然后，将 I^{down} 中第 j 组（j>1）各个物品依次替换为 I^{up} 的第 j-1 组物品；最后，将 I^{up} 中每个剩余物品（至多 $n\varepsilon^2$ 个）放入一个新的箱子。因此，$Opt(I^{up})\le Opt(I^{down})+n\varepsilon^2$。于是，由(3)可得，$Opt(I^{up})\le Opt(I^{down})+n\varepsilon^2\le Opt(I')+\varepsilon Opt(I')=(1+\varepsilon)Opt(I')$。 □

体积种类为固定值的装箱问题动态规划算法

由于问题实例中物品体积 $a_1,a_2,\ldots,a_n$ 仅有 K 种不同取值 $s_1<s_2...<s_K$，因此该实例可以表示为一个 K-元组 $\langle i_1,i_2,...,i_K\rangle$，其中 i_j（$1\le j\le K$）表示实例中体积为 s_j 的物品共有 i_j 个,并且 $n=\sum_{j=1}^{K} i_j$ 。

令 $Bin(i_1,i_2,...,i_K)$表示在问题实例 $\langle i_1,i_2,...,i_K\rangle$ 所需箱子的最小个数。

为了使用动态规划算法求解实例 $\langle n_1,n_2,...,n_K\rangle$,其中 $n=\sum_{j=1}^{K} n_j$ 。首先，计算集合

$$Q=\{\langle i_1,i_2,...,i_K\rangle \mid 0\le i_j\le n_j \text{ 对 } 0\le j\le K \text{ 成立且 } Bin(i_1,i_2,...,i_K)=1\}$$

由于满足“$0\le i_j\le n_j$ 对 $0\le j\le K$ 成立”的 K-元组 $\langle i_1,i_2,...,i_K\rangle$ 至多有 n^K 个，并且验证 $Bin(i_1,i_2,...,i_K)=1$（亦即 $\langle i_1,i_2,...,i_K\rangle$ 中非零分量个数是否等于 1）是否成立仅需 $O(K)$ 时间，因此计算 Q 的时间复杂度为 $O(Kn^K)$。

然后，利用递推式（或优化子结构）

$$Bin(i_1,i_2,...,i_K)=1+min\ \{Bin(i_1-q_1,i_2-q_2,...,i_K-q_K) \mid \ \langle q_1,q_2,...,q_K\rangle\in Q\}$$

自底向上地计算 $Bin\langle n_1,n_2,...,n_K\rangle$。由于需要利用递归式求解的子问题 $\langle i_1,i_2,...,i_K\rangle$ 至多有 n^K 个，且计算 $Bin(i_1,i_2,...,i_K)$时遍历 Q 所需的时间复杂度为 $O(n^K)$，因此在(n^{2K})时间内可以求得问题实例 $\langle n_1,n_2,...,n_K\rangle$ 的优化解的代价。

利用计算得到的 K-维数组 $Bin(i_1,i_2,...,i_K)$，其中 $0\le i_j\le n_j$ 对 $0\le j\le K$ 成立，容易在 $O(n)$时间内构造出问题实例 $\langle n_1,n_2,...,n_K\rangle$ 的最优解 S,其中 $n=\sum_{j=1}^{K} n_j$ 。

将上述算法称为算法 DynamicSearch(I,K)，其总时间复杂度为(n^{2K})。

将转换后实例的优化解转换为原问题的近似解

引理 10.22 表明，算法 DynamicSearch(I^{up},$1/\varepsilon^2$)输出的 I^{up} 的优化解 S 可以作为

I'的一个近似解。利用贪心策略可以将 I'的近似解 S 转换成原问题实例 I 的近似解。转换过程依次考察 I 中体积小于 ε 的每个物品 i，如果 S 中存在一个可以继续容纳物品 i 的箱子，则将物品 i 装入该箱子；否则，开启新箱子并将物品 i 装入，将更新后的装箱方案仍记为 S。显然，这一过程在 $O(n)$时间内得到 I 的一个可行解。

近似解转换算法 SolutionTrans(S,I,ε)
输入：装箱问题的实例 I，I 中体积大于 ε 的物品的一个近似装箱方案 S
输出：I 的一个近似解
1. For I 中体积小于 ε 的每个物品 i Do
2. If S 中存在箱子能够容纳物品 i Then 将物品 i 装入该箱子；
3. Else 开启新箱子并将物品 i 装入，将更新后的装箱方案仍记为 S;
4. 输出更新后的装箱方案 S;

算法分析

定理 10.23.给定 $\varepsilon\in(0,1/2)$和装箱问题实例 I,其中$|I|=n$,则算法 ApproxBinPacking(I,ε)在 $O(n^{2/\varepsilon^2})$时间内求得 I 的 $1+2\varepsilon$-近似解。

证明. 根据前面对算法第 1-3 步的操作过程和时间复杂度的讨论可知，算法在 $O(n^{2/\varepsilon^2})$时间内求得问题的一个近似解。仅需证明该近似解的近似比为 $1+2\varepsilon$。

为此，用 $approx(I)$表示 ApproxBinPacking(I,ε)输出的近似解使用的箱子的个数，用 $Opt(I),Opt(I')$和 $Opt(I^{up})$分别表示 I,I'和 I^{up} 的最优解使用的箱子的个数，用 new 表示算法 SolutionTrans(S,I,ε)第 3 步新开箱子的个数，则由算法操作过程和引理 10.22 可知

$$Opt(I')\le Opt(I)$$
$$Opt(I^{up})\le(1+\varepsilon)Opt(I')$$
$$approx(I)= Opt(I^{up})+new$$
$$new\le Opt(I)$$

如果 new=0,则 $approx(I)= Opt(I^{up})\le(1+\varepsilon)Opt(I')\le(1+2\varepsilon)Opt(I)$，定理成立。

否则，$new\ge1$。这表明 ApproxBinPacking(I,ε)输出的近似解使用的 $approx(I)$个箱子中，除最后一个箱子之外，每个箱子的剩余空间均不超过 ε,因此这些箱子内物品的总体积至少为$(1-\varepsilon)\cdot[approx(I)-1]$。它小于等于 I 中所有物品的总体积，且实例 I 的优化解使用的箱子个数至少为物品的总体积。于是 $(1-\varepsilon)\cdot[approx(I)-1]\le Opt(I)$。进而

$$\begin{aligned} approx(I) &\le \frac{1}{1-\varepsilon}Opt(I)+1 \\ &\le (1+2\varepsilon)Opt(I)+1 \end{aligned}$$

□

10.6 基于线性规划的近似算法

许多组合优化问题可以表示成(整数)线性规划问题。线性规划问题具有优美的对偶性质，并且在多项式时间内可解。这使得线性规划方法在近似算法设计和分析过程中扮演着非常重要的作用。事实上，许多难解的组合优化问题的近似算法均基于该方法设计得到。本节介绍线性规划方法在近似算法设计中的应用。

10.6.1 线性规划及对偶定理

线性规划是在决策变量的一组线性约束下求解线性目标函数的最小值，其对偶问题要求在相应的一组线性约束下求解相应的线性目标函数的最大值。如图 10.5 所示，线性规划问题及其对偶可以紧凑地表示为矩阵形式,其中 b, c 分别是 m 维和 n 维实数列向量，A 是 $m\times n$ 的实数矩阵, x 和 y 分别是 n 维和 m 维实变量列向量, b^T, c^T 和 A^T 分别表示 b, c 和 A 的转置。

$min \quad c^Tx$	$max \quad b^Ty$
$s.t. \quad A_{m\times n}x\geq b$	$s.t. \quad A^Ty\leq c$
$x\geq 0$	$y\geq 0$
(a) 线性规划问题	(*b*) 线性规划问题的对偶问题

图 10.5 线性规划问题及其对偶

为理解线性规划问题及其对偶问题之间的对偶关系，考虑如下所示的实例。

线性规划实例	实例的对偶
$min\ 7x_1+x_2+5x_3$	$max\ 10y_1+6y_2$
$s.t. \quad x_1-x_2+3x_3\geq 10$	$s.t. \quad y_1+5y_2 \leq 7$
$5x_1+2x_2-x_3\geq 6$	$-y_1+2y_2\leq 1$
$x_1,x_2,x_3\geq 0$	$3y_1-y_2 \leq 5$
	$y_1,y_2\geq 0$

对上述实例的第 1 个约束条件和第 2 个约束条件进行非负线性组合（目的是使得不等式方向保持不变），有如下推导

$$\begin{aligned}10y_1+6y_2 &\leq y_1(x_1-x_2+3x_3)+y_2(5x_1+2x_2-x_3)\\ &=(y_1+5y_2)x_1+(-y_1+2y_2)x_2+(3y_1-y_2)x_3\\ &\leq 7x_1+x_2+5x_3\end{aligned}$$

注意，最后的不等关系成立的条件是“$y_1+5y_2 \leq 7$ 且$-y_1+2y_2\leq 1$ 且 $3y_1-y_2\leq 5$”，这正是该实例的对偶中给出的约束条件。这说明，该实例的目标函数取值绝不可能小于其对偶的目标函数取值；反之，对偶的目标函数取值绝不可能大于该实例的目标函数取值。容易验证，可行解 $x=(7/4,0,11/4)^T$ 和 $y=(2,1)^T$ 使得两个目标函数均取值为 26。由此断言，$x=(7/4,0,11/4)^T$ 和 $y=(2,1)^T$ 分别是实例及其对偶的优化解。

上面观察到的结论即线性规划的对偶性。在此，我们叙述这些结论但不做证明。

定理 10.24(线性规划对偶定理). 原问题的优化解取有限值当且仅当对偶问题的优化解取有限值。而且，如果 $x^*=(x_1^*,x_2^*,...,x_n^*)$ 和 $y^*=(y_1^*,y_2^*,...,y_m^*)$ 分别是原问题和对偶问题的优化解，则 $\sum_{j=1}^{n} c_j\cdot x_j^*=\sum_{i=1}^{m} b_i\cdot y_i^*$。 □

定理 10.25(弱对偶定理).设 $x=(x_1,x_2,...,x_n)$和 $y=(y_1,y_2,...,y_m)$分别是原问题和对偶问题的可行解，则 $\sum_{j=1}^{n} c_j\cdot x_j \geq \sum_{i=1}^{m} b_i\cdot y_i$。 □

定理 10.26(互补松弛条件).设 $x=(x_1,x_2,...,x_n)$和 $y=(y_1,y_2,...,y_m)$分别是原问题和对偶问题的可行解，则 x 和 y 均是优化解当且仅当下面的两个条件同时成立：

(1)原松弛条件： 要么 $x_j=0$ 要么 $\sum_{i=1}^{m} a_{ij}\cdot y_i = c_j$ 对于 $1\leq j\leq n$ 成立；

(2)对偶松弛条件：要么 $y_i=0$ 要么 $\sum_{j=1}^{n} a_{ij}\cdot x_j = b_i$ 对于 $1\leq i\leq m$ 成立。 □

由定理互补松弛条件容易得出如下的定理，它是一类近似算法设计的基础。

定理 10.27. 设 $x=(x_1,x_2,...,x_n)$和 $y=(y_1,y_2,...,y_m)$分别是原问题和对偶问题的可行解，且

(1)要么 $x_j=0$ 要么 $\frac{c_j}{\alpha} \le \sum_{i=1}^{m} a_{ij} \cdot y_i \le c_j$ 对于 $1 \le j \le n$ 成立；

(2)要么 $y_i=0$ 要么 $b_i \le \sum_{j=1}^{n} a_{ij} \cdot x_j \le \beta b_i$ 对于 $1 \le i \le m$ 成立；

则 $\sum_{j=1}^{n} c_j \cdot x_j \le \alpha \cdot \beta \cdot \sum_{i=1}^{m} b_i \cdot y_i$。

证明. $\sum_{j=1}^{n} c_j \cdot x_j \le \alpha \sum_{j=1}^{n} (\sum_{i=1}^{m} b_i \cdot y_i) \cdot x_j = \alpha \sum_{i=1}^{m} (\sum_{j=1}^{n} c_j \cdot x_j) \cdot y_i \le \alpha \cdot \beta \cdot \sum_{i=1}^{m} b_i \cdot y_i$。

定理 10.28.线性规划问题可以在多项式时间内求解。 □

10.6.2 加权集合覆盖问题的线性规划表示

集合覆盖问题的输入是 n-元集合 X 的一个子集族 $\boldsymbol{F}=\{S_1,S_2,...,S_m\}$，其中 $X=\cup_{S\in\boldsymbol{F}}S$ 且 $\forall S\in\boldsymbol{F}$ 具有非负权值 $w(S)$，输出是子集族 $\boldsymbol{C}\subseteq F$ 使 $X=\cup_{S\in\boldsymbol{C}}S$ 且 $\sum_{S\in C} w(S)$ 达到最小。

加权集合覆盖问题可以表示为图 10.6(a)所示的整数线性规划 ***ILP***。事实上，对于 $\forall S\in\boldsymbol{F}$，用 $x_S=1$ 表示 S 在输出集族 $\boldsymbol{C}$ 中，用 $x_S=0$ 表示 S 不在输出集族 $\boldsymbol{C}$ 中。因此，输出集族的总权值 $\sum_{S\in C} w(S)$ 最小亦即线性函数 $\sum_{S\in F} w(S)\cdot x_S$ 达到最小值。同时，问题的约束 $X=\cup_{S\in\boldsymbol{C}}S$ 则意味着，对于 $\forall e\in\boldsymbol{X}$，集族 $\boldsymbol{F}$ 中至少有一个包含元素 e 的子集位于输出子集族 $\boldsymbol{C}$，即 $\sum_{e\in S} x_S \ge 1$ 对 $\forall e\in\boldsymbol{X}$ 成立。

ILP 与加权集合覆盖等价，它仍然是 *NP*-完全问题。将 ***ILP*** 中变量的取值范围 $x_S\in\{0,1\}$ 松弛为 $x_S\in[0,1]$ 得到图 10.6(b)所示的线性规划 ***LP***。由定理 10.28 可知，***LP*** 可以在多项式时间内求得优化解 x^*。如果将 ***LP*** 的约束条件“$\sum_{e\in S} x_S \ge 1$ 对 $\forall e\in\boldsymbol{X}$ 成立”用矩阵形式表示为 $Ax\ge\mathbf{1}$，则 A 的第 i 行标识 $\boldsymbol{F}$ 中的一个集合 S_i，第 j 列标识 X 中的一个元素 e_j,且 $a_{ij}=1$ 当且仅当 $e_j\in S_i$。于是，A^T 的第 j 行标识 X 中的一个元素 e_j，第 i 列标识 $\boldsymbol{F}$ 中的一个集合 S_i，且 $a_{ij}=1$ 当且仅当 $e_j\in S_i$。因此，“$A^Ty\le\boldsymbol{c}$”意味着“$\sum_{e\in S} y_e \le w(S)$ 对 $\forall S\in\boldsymbol{F}$ 成立”。由此，得到 10.6(c)所示的对偶线性规划问题 ***DLP***。

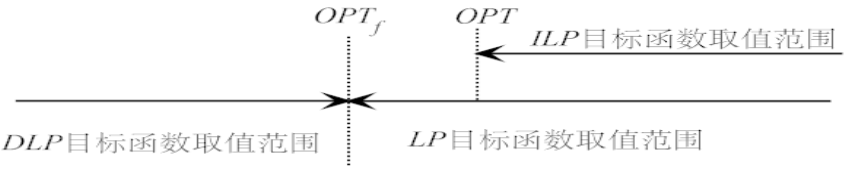
$min \ \sum_{S\in F} w(S)\cdot x_S$	$min \ \sum_{S\in F} w(S)\cdot x_S$	$max \ \sum_{e\in X} y_e$
$s.t. \ \sum_{S\in F:e\in S} x_S \ge 1 \quad \forall e\in\boldsymbol{X}$	$s.t \ \sum_{S\in F:e\in S} x_S \ge 1 \quad \forall e\in\boldsymbol{X}$	$s.t. \ \sum_{e\in S} y_e \le w(S) \quad \forall S\in\boldsymbol{F}$
$x_S\in\{0,1\} \quad \forall S\in\boldsymbol{F}$	$x_S\in[0,1] \quad \forall S\in\boldsymbol{F}$	$y_e\ge 0 \quad \forall e\in\boldsymbol{X}$
(a)整数线性规划 ***ILP***	(b)线性规划 ***LP***	(c)对偶线性规划 ***DLP***

图 10.6 加权集合覆盖问题的线性规划表示

将 ***ILP*** 和 ***LP*** 的目标函数的最小值分别为 OPT,OPT_f。显然，$0\le OPT_f \le OPT$ 并且 $OPT\le\sum_{S\in F} w(S)<+\infty$。因此，由定理 10.24 可得如下定理。

定理 10.29.***LP***,***DLP*** 的目标函数最优值是相同的有限值 OPT_f，并且不大于 ***ILP*** 的目标函数最小值 OPT,亦即 $0\le OPT_f\le OPT$。 □

下面的图直观地表示了三个规划问题的目标函数最优取值之间的大小关系。



定理 10.29 描述了能够用整数线性规划描述的组合优化问题的一般性质，它是基于线性规划求解组合优化问题的基础，从不同角度应用该结论将得到不同的近似算法。后续的小节，将以加权集合覆盖问题为例，介绍不同的算法设计策略。

10.6.3 舍入法

舍入法

基于线性规划设计近似算法的第一种策略是舍入法。该方法首先调用线性规划算法求得 ***LP*** 的优化解 x，然后用舍入方法将 x 的分量转换成整数得到 ***ILP*** 的一个可行解 x'，并将 x'作为问题的近似解输出。舍入法的关键在于，采用恰当的舍入策略确保舍入得到的整数向量 x'是 *ILP* 的可行解。分析近似比时，利用定理 10.29。

以加权集合覆盖问题为例，考虑如下的舍入近似算法 SetCoverRounding。算法先调用线性规划算法在多项式时间内求得 *LP* 问题的优化解 x。然后，对 x 的每个分量 x_S 进行舍入：如果 $x_S>1/f$，则令 $x_S=1$（将集合 S 添加到输出集族）；否则，令 $x_S=0$（舍弃集合 S），其中 f 是 X 的元素在集族 ***F*** 中的最大频率。显然，算法的复杂性是多项式时间。

算法 SetCoverRounding

1. 多项式时间内求得 ***LP*** 的优化解 x;
2. $f \leftarrow max_{e\in X}|\{S \mid S\in F$ 且 $e\in S\}|$　　/*f 是元素在集族 ***F*** 中的最大频率*/
3. If　$x_S>1/f$　Then　$\boldsymbol{C}\leftarrow\boldsymbol{C}\cup\{S\}$;
4. 输出 ***C***

首先，断言 ***C*** 是集合覆盖问题的一个近似解。考察$\forall e\in X$。由于 e 的频率不超过 f,故 F 中至多有 f 个集合 S 满足 $e\in S$；由于 x 是 ***LP*** 的优化解，它必然满足 *LP* 对应于 e 的约束 $\sum_{S\in F:e\in S} x_S \geq 1$。因此，必存在 $S_i\in\boldsymbol{F}$ 使得 $e\in S_i$ 且 $x_{S_i}>1/f$。根据算法第 3 步，$S_i\in\boldsymbol{C}$。由 e 的任意性知道，$X=\cup_{S\in\boldsymbol{C}}S$，即 ***C*** 是问题的可行解。

下面的分析过程给出了算法的近似比。

$$
\begin{aligned}
w(\boldsymbol{C}) &= \sum_{S\in C} w(S) \\
&= f\cdot\sum_{S\in C}\frac{1}{f}\cdot w(S) \\
&\leq f\cdot[\sum_{S\in C} x_S\cdot w(S) + \sum_{S\in F-C} x_S\cdot w(S)] \qquad \text{(根据算法第 3 步)} \\
&= f\cdot OPT_f \qquad \text{(x 是 LP 的优化解)} \\
&\leq f\cdot OPT \qquad \text{(定理 10.29)}
\end{aligned}
$$

定理 10.30.算法 SetCoverRounding 是集合覆盖问题的 f-近似算法，其中 $f=max_{e\in X}|\{S \mid S\in F$ 且 $e\in S\}|$ 是 X 中元素在 ***F*** 中的最大频率。 □

随机舍入法

舍入法的另一种形式是随机舍入。该方法首先调用线性规划算法求得 ***LP*** 的优化解 x，然后用随机舍入方法，独立地以一定概率将 x 的各个分量转换成整数得到 y 一个整数向量 x'。x'并不一定是 ***ILP*** 的可行解。分析算法时，需要将随机舍入算法视为随机算法，分析问题获得问题近似解的概率和近似解的期望代价。

以加权集合覆盖问题为例，考虑如下的随机舍入近似算法 SetCoverRRounding。算法先调用线性规划算法在多项式时间内求得 *LP* 问题的优化解 x。根据 ***LP*** 的定义，$0 \le x_S \le 1$ 对 $\forall S \in \boldsymbol{F}$ 成立。算法将每个 x_S 视为一个概率值，独立地将 x 的每个分量 x_S 进行随机舍入，确保 $x_S^{'}=1$ 的概率为 x_S。显然，算法的时间复杂度是多项式时间。

算法 SetCoverRRounding

1.多项式时间内求得 ***LP*** 的优化解 x;

2.For $\forall S \in \boldsymbol{F}$ **Do**

3. 产生[0,1]区间内随机数 *rand*;

4. If *rand*>1-x_S Then $\boldsymbol{C} \leftarrow \boldsymbol{C} \cup \{S\}$; /*$S \in \boldsymbol{C}$ 的概率为 x_S*/

5. 输出 ***C***

定理 10.31.算法 SetCoverRRounding 的输出集合为 C 满足

(1) $E(w(C))=OPT_f$;

(2)X 中任意元素被 ***C*** 覆盖的概率大于 1-1/e，其中 e 是自然对数底数。

证明. 先证(1)。 $E(w(C))=\sum_{S \in F} \Pr(S \in C) \cdot w(S) = \sum_{S \in F} x_S \cdot w(S) = OPT_f$ 。

往证(2)。$\forall a \in X$，设 ***F*** 中恰有 k 个集合 S 包含元素 a，不妨设这些集合在 ***LP*** 中对应变量 $x_1,x_2,...,x_k$。于是，***LP*** 的优化解 x 满足

$$x_1+x_2+...+x_k \ge 1$$

于是，

Pr[a 未被 ***C*** 覆盖] = Pr[包含 a 的 k 个集合均未被选入 ***C***]

$$= (1-x_1)(1-x_2)\cdots(1-x_k)$$

$$\le \left(1-\frac{x_1+x_2+...+x_k}{k}\right)^k$$

$$\le \left(1-\frac{1}{k}\right)^k$$

Pr[a 被 ***C*** 覆盖]=1- Pr[a 未被 ***C*** 覆盖]≥ 1-(1-1/k)^k ≥1-1/e。 □

习题 10.7 讨论了随机舍入算法的改进。

10.6.4 对偶拟合方法（Dual Fitting method）

分析组合优化近似算法的近似比时，难点在于将近似解的代价和优化解的代价用不等式关联起来。前面的例子表明，这种分析通常需要根据问题和算法的特征采用特殊的关联手段，不具有通用性。然而，对于能够表示成整数线性规划的组合优化问题，对偶拟合方法为组合优化近似算法的近似比分析提供了比较通用的方法。

对偶拟合的基本思想是利用定理 10.29 建立近似解代价和优化解代价的不等式约束关系。以最小化组合优化问题为例，组合优化近似算法给出的近似解的代价 C 和问题优化解的代价 OPT 满足 $OPT \le C$。根据定理 10.29 可知，$OPT_f \le OPT \le C$，其中 OPT_f

是 ***LP*** 和 ***DLP*** 的（共同的）优化解代价。对偶拟合方法的思想是，在设计组合优化近似算法计算问题的近似解时，想办法确保能够利用近似解和缩放因子α来“拟合”对偶问题(即 ***DLP***)的可行解，则可以建立不等式$\alpha^{-1}\cdot C\le OPT_f\le OPT$。由此可知，近似算法的近似比为$\alpha$。

下面以加权集合覆盖问题为例，说明对偶拟合方法的原理。考虑如下求解加权集合覆盖问题的组合优化算法，它是 10.3.1 节中贪心算法的直接推广。由于选择集合 *S* 添加到输出集族 ***C*** 中时，近似解的代价将增加 *w*(*S*)，而首次被 *S* 覆盖的元素构成集合 *S*-*U*,其中 *U* 记录了目前已被覆盖的所有元素。为了使得每个元素被首次覆盖时的平均代价最小，算法总贪心地选择 *w*(*S*)/|*S*-*U*|达到最小值的集合 *S* 添加到 ***C*** 中。后面将证明，每个元素分得的平均代价 y_e 经过适当缩放将构成（或“拟合”）***DLP*** 的近似解，这使得算法的近似比分析变得更容易。

加权集合覆盖对偶过滤算法 SetCoverDualFitting

1. *U*←∅, ***C***←∅; /**U* 记录已被 ***C*** 覆盖的所有元素*/
2. while *U*≠*X* do
3. 从 ***F*** 中挑选一个集合 *S* 使得 *w*(*S*)/|*S*-*U*|最小;
4. 对∀e∈*S*-*U*，令 y_e=*w*(*S*)/|S-*U*|;
5. ***C***←***C***∪{*S*}，*U*←*U*∪*S*, ***F***←***F***-{*S*};
6. 输出 ***C***

引理 10.32. 算法 SetCoverDualFitting 的输出集族 ***C*** 是加权集合覆盖问题的近似解，且 $w(\boldsymbol{C})=\sum_{S\in C} w(S)=\sum_{e\in X} y_e$ 。

证明.不妨设算法依次选入集族 ***C*** 的集合是 $S_1,S_2,...,S_{|C|}$,则根据算法第 2 步的循环终止条件可知，$X=\cup_{S\in C}S$。因此，***C*** 是加权集合覆盖问题的可行解。而且，

$$\begin{aligned}\sum_{e\in X} y_e &= \sum_{e\in S_1} y_e+\sum_{e\in S_2-S_1} y_e+...+\sum_{e\in S_{|C|}-(S_1\cup...\cup S_{|C|-1}} y_e\\ &= |S_1|\cdot\frac{w(S_1)}{|S_1|}+|S_2-S_1|\cdot\frac{w(S_2)}{|S_2-S_1|}+...+|S_{|C|}-(S_1\cup...\cup S_{|C|})|\cdot\frac{w(S_{|C|})}{|S_{|C|}-(S_1\cup...\cup S_{|C|})|}\\ &=\sum_{S\in C} w(S)\\ &=w(\boldsymbol{C})\end{aligned}$$ □

引理 10.33.算法 SetCoverDualFitting 中定义的 y_e 具有如下性质，由所有 $y_e/H(n)$构成的列向量是 ***DLP*** 的一个可行解，其中 $H(n)=\sum_{j=1}^{n}\frac{1}{j}$。

证明. 显然, $y_e/H(n)\ge 0$ 对任意 $e\in X$ 成立，故只需对任意 $S\in\boldsymbol{F}$ 验证 $\sum_{e\in S}y_e/H(n)\le w(S)$ 成立。为方便讨论，不妨设将 *S* 中所有元素依它们被算法选中的集合覆盖的先后次序列出为 $e_1,\ldots,e_k$。

考察 e_i 被算法选中的集合首次覆盖的时刻，由于 *S* 中此时还有 *k*-*i*+1 个元素未被覆盖，将 *w*(*S*)平摊到这些元素上，每个元素将分得代价 *w*(*S*)/(*k*-*i*+1)。

由于在 e_i 首次被覆盖时，*S* 本身也是侯选集合，且选择集合 *S*′时要求 *w*(*S*′)/|*S*′-*U*|达到最小，故 $y_{e_i}\le c(S)/(k-i+1)$。进而

$$\frac{y_{e_i}}{H(n)}\le\frac{1}{H(n)}\cdot\frac{w(S)}{k-i+1}$$

于是，

$$\sum_{e\in S}\frac{y_e}{H(n)}=\sum_{i=1}^{k}\frac{y_{e_i}}{H(n)}\le\frac{w(S)}{H(n)}\cdot\sum_{i=1}^{k}\frac{1}{k-i+1}=\frac{w(S)}{H(n)}\cdot H(k)\le w(S)$$ □

定理 10.34.算法 SetCoverDualFitting 的近似比为 ln *n*+1。

证明. 由引理 10.33,定理 10.24 和定理 10.29 可知

$$\sum_{e\in X} y_e / H(n)\le OPT_f\le OPT$$

再由引理 10.32 和 $H(n)$的阶可知

$$w(\boldsymbol{C})\le H(n)\cdot OPT\le(\ln n+1)\,OPT$$ □

10.6.5 原偶模式（Primal Dual Schema）

原偶模式利用定理 10.27 设计近似算法，由此得到的近似算法一般具有复杂性低和近似比小的优点。原偶模式从原始问题 ***ILP*** 的非可行解 *x* 和对偶问题 ***DLP*** 的可行解 *y* 出发(一般将 *x* 和 *y* 均初始化为向量 **0**)。然后，利用恰当的策略逐步将 *x* 改造成原问题的可行解，同时改进对偶问题可行解 *y* 使得目标函数取值逐渐接近最优值 OPT_f。特别值得注意的是，对 *x* 和 *y* 的改造过程是交互的，亦即 *x* 的每一次改造将触发对 *y* 的新的改造；反之，对 *y* 的每一次改造也将触发对 *x* 的改造。改造过程结束后，应确保定理 10.27 中的两个条件对恰当的α和β成立。此时，由定理 10.27 和定理 10.29 可知，算法的近似比为$\alpha\cdot\beta$。

应用原偶模式时，通常的做法是固定α（或β）等于 1，然后在算法设计过程中选择恰当β（或α）以确保 “$y_i\neq0\Rightarrow b_i\le\sum_{j=1}^{n}a_{ij}\cdot x_j\le\beta b_i$ 对于 $1\le i\le m$ 成立”（或“$x_j\neq0\Rightarrow\frac{c_j}{\alpha}\le\sum_{i=1}^{m}a_{ij}\cdot y_i\le c_j$ 对于 $1\le j\le n$ 成立”）。β（或α）的选取依赖于具体问题的特征。

下面，以加权集合覆盖问题为例说明用原偶模式设计近似算法的过程。

加权集合覆盖原偶模式算法 SetCoverPrimalDual

1. $x\leftarrow0$; /*$\forall S\in F$ 在 x 中对应分量 x_S*/
2. $y\leftarrow0$; /*$\forall e\in X$ 在 y 中对应分量 y_e*/
3. $\boldsymbol{U}\leftarrow\varnothing$; $\boldsymbol{H}\leftarrow\boldsymbol{F}$; /*$U$ 记录 X 中已被覆盖的元素*/
4. While $U\neq X$ Do
5. 选择 $e_0\in X-U$;
6. 增大 y_{e_0} 直到 ***H*** 中某个满足 $y_{e_0}\in S$ 的集合上 $\sum_{e\in S}y_e=w(S)$ 成立；
7. $x_S\leftarrow1$; $\boldsymbol{U}\leftarrow U\cup S$; $\boldsymbol{H}\leftarrow\boldsymbol{H}-\{S\}$；
8. 输出集族 $\boldsymbol{C}=\{S\mid S\in\mathbf{F}$ 且 $x_S=1\}$

引理 10.35. 算法 SetCoverPrimalDual 结束后，*x* 和 *y* 分别是 ***ILP*** 和 ***DLP*** 的可行解。

证明. 先证明 *x* 在算法结束后是 ***ILP*** 的可行解。根据算法第 7 步，只有当元素被满足 $x_S=1$ 的集合 *S* 覆盖之后，才被标记为已被覆盖(即放入 *U* 中)。再由第 4 步的条件，只有当 *X* 中所有元素被覆盖后，算法才结束。因此，算法结束后，*X* 中每个元素至少被满足 $x_S=1$ 的一个集合 *S* 覆盖，即 *x* 是 ***ILP*** 的可行解。

下面证明算法结束后，*y* 是 ***DLP*** 的可行解。注意，算法开始时，*y*=**0** 是 ***DLP***

的可行解，即 $\sum_{e\in S} y_e \le w(S)$ 对 $\forall S\in \boldsymbol{F}$ 成立。一方面，算法每次在第 5 步和第 6 步根据未被覆盖的元素 e_0 选择集合 S 后，集合 S 上的约束 $\sum_{e\in S} y_e = w(S)$ 成立。此后，S 中所有元素被标记为已被覆盖，故 $\forall e\in S$ 的分量 y_e 在后续步骤中不会再增大。因此，$\sum_{e\in S} y_e = w(S)$ 将维持到算法结束。另一方面，上述分析同时表明，在未被算法第 5 步和第 6 步选中的集合 S 上 $\sum_{e\in S} y_e \le w(S)$ 将一直保持到算法结束。 □

引理 10.36. 算法 SetCoverPrimalDual 结束后，x 和 y 满足如下条件：

(1)对于 $\forall S\in \boldsymbol{F}$，$x_S\neq 0 \Rightarrow \sum_{e\in S} y_e = w(S)$；

(2)对于 $\forall e\in X$，$y_e\neq 0 \Rightarrow \sum_{S\in F: e\in S} x_S \le f$，其中 $f=max_{e\in X}|\{S \mid S\in F \text{ 且 } e\in S\}|$ 是元素在集族 $\boldsymbol{F}$ 中的最大频率。

证明. 由算法第 6 步和第 7 步可知(1)成立。

根据 f 的定义，$\forall e\in X$,在 $\boldsymbol{F}$ 中至多存在 f 个集合 S 使得 $e\in S$；而且 x_S=1 或 x_S=0；故(2)成立。 □

定理 10.37. 算法 SetCoverPrimalDual 的近似比为 f。

证明. 由引理 10.35，引理 10.36，定理 10.27 和定理 10.29 可得如下推导

$$
\begin{aligned}
w(\boldsymbol{C}) &= \sum_{S\in C} w(S) && (w(\boldsymbol{C})\text{的定义})\\
&= \sum_{S\in F} w(S)\cdot x_S && (x_S=1\Rightarrow S\in\boldsymbol{C}\text{，算法第 7 步})\\
&\le f\cdot \sum_{e\in X} y_e && (\text{引理 10.35，引理 10.36，定理 10.27})\\
&\le f\cdot OPT_f && (y \text{ 是 } \boldsymbol{DLP} \text{ 的可行解})\\
&\le f\cdot OPT && (\text{定理 10.29})
\end{aligned}
$$

□

10.7 不可近似性

10.2-10.6 节借助不同的技术为许多 *NP*-完全问题设计了近似算法。这些近似算法中有的是完全多项式近似模式，还有的具有常数近似比，另有一些的近似比形如 $\log n$，$2\sqrt{m}$ 等等。为什么不为每个计算问题都建立完全多项式近似模式，或者建立近似比为常数的近似算法呢？这是由于对某些 *NP*-完全问题设计这样的近似算法同证明“*P=NP*”一样难。因此，在 *P≠NP* 的假设下，可以证明特定的近似算法不存在；这种结论称为问题的不可近似性。

本节初步地讨论建立不可近似性的相关技术。10.7.1 小节讨论利用α-鸿沟归约建立不可近似性；10.7.2-10.7.3 小节讨论建立α-鸿沟归约的一般技术；10.7.4 小节讨论利用已有不可近似性建立新的不可近似性的技术。

10.7.1 鸿沟归约与不可近似性

定义 10.8. （***α*-鸿沟归约**）设 α>1 是常数，问题 *A* 是一个判定性 *NP*-完全问题，问题 *B* 是一个最小值问题（或最大值问题）且问题 *B* 在实例 I_B 上的优化解的目标函数

值为 $Obj(I_B)$。问题 A 可以 **α-鸿沟归约**到问题 B，是指存在多项式时间算法 f 和多项式时间可计算的实数值 $W(\alpha,I_B)$满足:(1)f 将问题 A 的任意实例 I_A 转换成问题 B 的一个实例 I_B。(2)I_A 的解为“yes”当且仅当 $Obj(I_B)\le W(\alpha,I_B)$（或 $Obj(I_B)\ge W(\alpha,I_B)$）;$I_A$ 的解为“no”当且仅当 $Obj(I_B)\ge\alpha W(\alpha,I_B)$（或 $Obj(I_B)\le\alpha^{-1}\cdot W(\alpha,I_B)$）。

定理 10.38.（**鸿沟定理**）若一个判定性 *NP*-完全问题可以 α-鸿沟归约到一个优化问题，则该优化问题不存在多项式时间 α-近似算法，除非 *NP*=*P*。

证明.仅证明最小值问题的鸿沟定理，最大值问题的鸿沟定理证明类似（留给读者）。

利用反证法证明鸿沟定理。如果判定性 *NP*-完全问题 A 可以 α-鸿沟归约到最小值问题 B，且问题 B 存在一个多项式时间 α-近似算法 $\mathcal{B}$，下面证明问题 A 存在多项式算法 $\mathcal{A}$。这与问题 A 是 *NP*-完全问题矛盾，除非 *NP*=*P*，从而定理成立。

设 α-鸿沟归约中的多项式算法为 f，则问题 A 存在如下算法 $\mathcal{A}$。显然，它是多项式时间算法，因为每个计算步骤均仅需多项式时间。

求解问题 A 的算法 $\mathcal{A}$

输入：问题 A 的实例 I_A

输出：实例 I_A 是否有解

1. 利用 α-鸿沟归约中的多项式算法 f 将 I_A 转换成问题 B 的实例 I_B;
2. 在多项式时间内计算实数值 $W(\alpha,I_B)$;
3. 利用问题 B 的 α-近似算法 $\mathcal{B}$ 求得实例 I_B 的近似解 s_B 及其代价 $cost(S_B)$;
4. 若 $cost(S_B)\le\alpha W(\alpha,I_B)$，则输出“yes”;否则，输出“no”

算法 $\mathcal{A}$ 是正确的。事实上，由于算法 $\mathcal{B}$ 是一个 α-近似算法，故 $cost(S_B)\le\alpha\cdot Opt(I_B)$，其中 $Opt(I_B)$是实例 I_B 的优化解的代价。因此，算法 $\mathcal{A}$ 中第 4 步“$cost(S_B)\le\alpha W(\alpha,I_B)$”成立当且仅当“$Opt(I_B)\le W(\alpha,I_B)$”成立。再由 α-鸿沟归约的定义可知，“$Opt(I_B)\le W(\alpha,I_B)$”当且仅当“$I_A$ 的解为‘yes’”。因此,算法输出“yes”当且仅当 I_A 的解为“yes”。 □

利用鸿沟定理，可以证明 10.2-10.6 节中遇到的一些问题的不可近似性。

定理 10.39.装箱问题不存在 3/2-近似算法。

证明.如下的集合划分问题可以 3/2-鸿沟归约到装箱问题。由于集合划分问题是 *NP*-完全问题，故定理 10.38 表明，装箱问题不存在 3/2-近似算法，除非 *P*=*NP*。

集合划分问题

输入：正整数集合 $X=\{x_1,x_2,...,x_n\}$

输出：是否存在 $Y\subseteq X$ 使得 $\Sigma_{x_i\in Y}x_i=\Sigma_{x_i\in X-Y}x_i$

假设集合划分问题的一个实例 $X=\{x_1,x_2,...,x_n\}$满足 $x_i\le\frac{1}{2}\sum_{i=1}^{n}x_i$ 对 $1\le i\le n$ 成立，否则可直接判定集合划分问题的解为“no”。

给定集合划分问题的一个实例 $X=\{x_1,x_2,...,x_n\}$,令 $a=\frac{1}{2}\sum_{i=1}^{n}x_i$ 且 $a_i=x_i/a$，其中 $1\le i\le n$。易知，$a_i\in(0,1]$。于是，得到装箱问题的实例 $B=\{a_1,...,a_n\}$。而且，由 X 构造 B 可以在 $\Theta(n)$时间内完成。

显然,集合划分问题的实例 X 的解为“yes”当且仅当装箱问题的实例 $B=\{a_1,...,a_n\}$ 的优化解恰好使用 2 个(容积为 1 的)箱子。集合划分问题的实例 X 的解为“no”当

且仅当装箱问题的实例 $B=\{a_1,...,a_n\}$的优化解至少使用 $3=\frac{3}{2}\cdot 2$ 个(容积为 1 的)箱子。

这表明，集合划分问题可以 3/2-鸿沟归约到装箱问题。 □

定理 10.40. 不满足三角不等式的旅行商问题不存在以常数为近似比的近似算法，除非 $P=NP$。

证明：由鸿沟定理，只需证明对任意常数 $\alpha>1$,哈密顿环问题可以 α-鸿沟归约到不满足三角不等式的旅行商问题。 哈密顿环问题是一个判定性 *NP*-完全问题。

哈密顿环问题

输入：图 $G=(V,E)$

输出：G 中是否存在哈密顿环

给定哈密顿环问题的实例 $G=(V,E)$,令 $G'=(V,E')$是顶点集 V 上的完全图，并定义 $\forall(u,v)\in E'$的权值 $c(u,v)$为

$$c(u,v)=\begin{cases}1 & 若(u,v)\in E\subseteq E'\\ \alpha|V|+1 & 若(u,v)\in E'-E\end{cases}$$

加权图 $G'=(V,E')$是旅行商问题的一个实例。显然，由图 G 构造加权图 $G'=(V,E')$可以在 $\Theta(|V|^2)$时间内完成。

一方面，如果 G 中存在哈密顿环 H，则加权函数 c 为 H 中的每条边的分配代价 1，因此 G'中存在代价为$|V|$的哈密顿环，即旅行商问题的实例 G'的优化解的代价不超过$|V|$。

另一方面，如果 G 中不存在哈密顿环，则 G'中任意哈密顿环至少包含一条 E'-E 中的边,由于加权函数 c 为所有这样的边分配权值 $\alpha|V|+1$，因此 G'中任意哈密顿环的代价至少为$(\alpha|V|+1)+(|V|-1)>\alpha|V|$。

这表明，对任意常数 $\alpha>1$,哈密顿环问题可以 α-鸿沟归约到不满足三角不等式的旅行商问题。 □

10.7.2 PCP 定理

建立 α-鸿沟归约的一般技术依赖于 *NP*-问题的概率性质。*NP*-问题最重要的概率性质由 PCP 定理给出，其中 PCP 指的是概率验证证明（Probability Checkable Proof）。

首先，用如下定义的 3SAT 问题为例说明概率验证证明的概念。

3SAT 问题

输入：n 个布尔变量 $x_1,...,x_n$ 上的 m 个析取表达式 E_j（$1\le j\le m$），其中 $E_j=x_{j1}\vee x_{j2}\vee x_{j3}$ 且 E_j 中的每个变量均是 $x_1,...,x_n$ 中某个变量或某个变量的非;

输出：是否存在变量 $x_1,...,x_n$ 的一个赋值使得 $E_1\wedge E_2\wedge...\wedge E_m$ 取值为真;

考虑如何验证变量 $x_1,...,x_n$ 的一个赋值 $\langle v_1,...,v_n\rangle$ 是否满足所有 E_j（$1\le j\le m$）。图 10.6 给出了 3SAT 问题的一个概率验证证明器（亦即一个蒙特卡罗算法），它随机产生一个位置 r（$1\le r\le m$）,然后从 $\langle v_1,...,v_n\rangle$ 中读取第 r 个析取表达式 E_r 中 3 个变量的值 v_{r1},v_{r2},v_{r3}。如果 v_{r1},v_{r2},v_{r3} 使得 E_r 取值为真，则输出“赋值 $\langle v_1,...,v_n\rangle$ 是问题实例的解”，否则，输出“问题实例无解”。显然，如果赋值 $\langle v_1,...,v_n\rangle$ 是 3SAT 问题实例的解，则概率验证证明器正确的概率为 1；如果 3SAT 问题实例的无解，则概率验证证明器在输入的任意赋值上的错误概率不超过 1-1/m。 1992 年，S.Arora 利用复杂

的过程，证明了 3SAT 问题存在错误概率小于 1/2 的概率验证证明器，进而得到 PCP 定理。

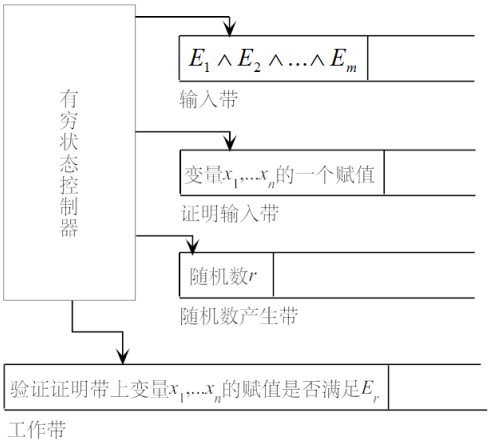


图 10.6 3SAT 问题的概率验证证明器

一般而言，*NP*-问题的一个概率验证证明器是一个多项式时间图灵机（亦即一个多项式时间蒙特卡罗算法），它由有穷状态控制器（总控程序）、输入带（存储输入数据 x 的空间）、证明输入带（存储解 y 的空间）、随机数产生带（产生随机数 r 的空间）和工作带（验证 y 是否为 x 的解的空间）构成。概率验证证明器能够在任意实例 x 和任意证明 y 上，读取 y 中由随机数 r 指定的位置上的数据，然后在工作带上进行计算并决定是否接受“y 是实例 x 的解”。

问题 $A\in$PCP(log$n$,1)是指，问题 A 存在一个概率验证证明器 V 和常数 c,q 使得 V 在问题 A 的任意实例 x 上满足：(1)V 使用的随机数的长度不超过 $c\log|x|$个二进制位,其中$|x|$表示实例 x 的输入规模;(2)V 根据随机数 r 恰好读取证明输入带上的 q 个位;(3)如果 y 是实例 x 的解，则（变换随机数 r 时）V 接受 y 的概率为 1；(4)如果实例 x 无解,则在任意证明 y 上（变换随机数 r 时）V 接受 y 的概率小于 1/2。

定理 10.41.（**PCP 定理**）*NP*=PCP(log$n$,1)。 □

PCP 定理断言，每个 *NP*-问题均存在多项式时间和错误概率小于 1/2 的概率验证证明器使得它仅使用问题输入规模的对数空间产生随机数并且仅访问证明中的常数个二进制位。如前所述，PCP 定理的证明过程即是设计 3SAT 问题的错误概率小于 1/2 的概率验证证明器，这是由于 3SAT 问题可以归约到任意的 *NP*-问题。PCP 定理的证明超出了本书讨论的范围。我们仅使用 PCP 定理来建立不可近似性相关的结论。

事实上，PCP 定理直接建立了如下极大值问题的不可近似性。

MAX-SAT 问题

输入：n 个布尔变量 $x_1,\ldots,x_n$ 上的 m 个析取表达式 E_j（$1\le j\le m$），其中 $E_j=x_{j1}\vee\ldots\vee x_{jk_j}$ 且 E_j 中的每个变量均是 $x_1,\ldots,x_n$ 中某个变量或某个变量的非;

输出：变量 $x_1,\ldots,x_n$ 的一个赋值使得 $E_1,E_2,\ldots,E_m$ 中被满足的表达式个数达到最大值

SAT 问题

输入：n 个布尔变量 $x_1,\ldots,x_n$ 上的 m 个析取表达式 E_j（$1\le j\le m$），其中 $E_j=x_{j1}\vee\ldots\vee x_{jk_j}$ 且 E_j 中的每个变量均是 $x_1,\ldots,x_n$ 中某个变量或某个变量的非;

输出：是否存在变量 $x_1,\ldots,x_n$ 的一个赋值满足布尔表达式 $E_1\wedge E_2\wedge\ldots\wedge E_m$

定理 10.42.MAX-SAT 问题不存在近似比为 2 的近似算法，除非 *NP=P*。

证明. 由于 SAT 问题是一个 *NP*-完全问题。由 PCP 定理可知，存在 SAT 问题的一个多项式时间的概率验证证明器 *V* 使得:(1)如果实例 $\phi=E_1\wedge E_2\wedge...\wedge E_m$ 是可满足的，则 *V* 在任意证明上输出“$E_1\wedge E_2\wedge...\wedge E_m$ 是可满足的”的概率为 1；(2)如果实例 $E_1\wedge E_2\wedge...\wedge E_m$ 是不可满足的，则 *V* 在任意证明上输出“$E_1\wedge E_2\wedge...\wedge E_m$ 是可满足的”的概率小于 1/2。

假设 MAX-SAT 问题存在近似比为 2 的多项式时间近似算法 $\mathcal{A}$。于是，对于可满足的布尔表达式 $E_1\wedge E_2\wedge...\wedge E_m$，算法 $\mathcal{A}$ 将输出变量 $x_1,...,x_n$ 的一个赋值 $\langle v_1,...,v_n\rangle$ 使得 $E_1,E_2,...,E_m$ 中至少有 *m*/2 个表达式被满足，亦即 *V* 接受证明 $\langle v_1,...,v_n\rangle$ 的概率大于等于 1/2。由于 *V* 中随机数的位数为 $\Theta(\log n)$,故不同随机数至多有 $\Theta(n)$个。因此，*V* 接受证明 $\langle v_1,...,v_n\rangle$ 的概率可以通过检查 $\Theta(n)$个数上 *V* 接受证明 $\langle v_1,...,v_n\rangle$ 的频率来计算，这只需多项式时间。这表明，算法 $\mathcal{A}$ 和验证证明器 *V* 结合，可以得到求解 SAT 问题的多项式时间算法，这不可能，除非 *NP=P*。 □

10.7.3 MAX-3SAT 问题的不可近似性

本小节利用 PCP 定理证明 MAX-3SAT 问题的不可近似性，其中 MAX-3SAT 问题是 3SAT 问题（见 10.7.2 节）的优化形式。 它在不可近似性理论中的作用类似于 SAT 问题在 *NP* 完全性理论中的作用，亦即所有的不可近似性结论均可以通过 MAX-3SAT 问题的不可近似性来获得。

MAX-3SAT 问题

输入：*n* 个布尔变量 $x_1,...,x_n$ 上的 *m* 个析取表达式 E_j（$1\le j\le m$），其中 $E_j=x_{j1}\vee x_{j2}\vee x_{j3}$ 且 E_j 中的每个变量均是 $x_1,...,x_n$ 中某个变量或某个变量的非;

输出：变量 $x_1,...,x_n$ 的一个赋值使得 $E_1,E_2,...,E_m$ 中被满足的表达式个数达到最大值

定理 10.43. 存在常数 $\varepsilon_M>0$,使得 MAX-3SAT 问题不存在近似比为$(1-\varepsilon_M)^{-1}$ 的近似算法。

证明.根据鸿沟定理（定理 10.38），只需将 SAT 问题$(1-\varepsilon_M)^{-1}$-鸿沟归约到 MAX-3SAT 问题，其中 SAT 问题是一个判定性的 *NP*-完全问题。

给定定义在布尔变量 $x_1,...,x_n$ 上的 SAT 问题的一个实例 I_{SAT}: $\phi=E_1\wedge E_2\wedge...\wedge E_m$,下面利用 PCP 定理在多项式时间内构造 MAX-3SAT 问题的一个实例 $\psi=F_1\wedge F_2\wedge...\wedge F_t$ 使得：(1)ϕ 是可满足的，当且仅当 ψ 是可满足的；(2)ϕ 是不可满足的，当且仅当 ψ 中至多$(1-\varepsilon_M)\cdot t$ 个析取表达式是可满足的，其中 *t* 是仅依赖于$|\phi|$，$\varepsilon_M>0$ 是常数。

由于 SAT 问题是一个 *NP*-完全问题，由 PCP 定理可知，存在 SAT 问题的一个概率验证证明器 *V* 和常数 *c*,*q* 使得 *V* 在 SAT 问题的任意实例 ϕ 上满足：(1)*V* 使用的随机数的长度不超过 $c\log|\phi|$个二进制位，其中$|\phi|$表示实例 ϕ 的二进制长度；(2)*V* 根据随机数 *r* 恰好读取证明输入带上的 *q* 个位;(3)如果 $\langle a_1,...,a_n\rangle$ 是实例 *x* 的解，则（变换随机数 *r* 时）*V* 接受 $\langle a_1,...,a_n\rangle$ 的概率为 1；(4)如果实例 ϕ 无解,则在任意证明 $\langle a_1,...,a_n\rangle$ 上（变换随机数 *r* 时）*V* 接受 $\langle a_1,...,a_n\rangle$ 的概率小于 1/2。

在实例 ϕ 上，由于 *V* 使用的随机数的长度不超过 $c\log|\phi|$个二进制位，故所有随机数 *r* 介于 1 到$|\phi|^c$ 之间，因此 *V* 可能访问证明输入带上至多 $q|\phi|^c$ 个二进制位。为这些二进制位中的每个位引入一个布尔变量 y_i （$1\le i\le q|\phi|^c$）,并为每个可能的随机数 *r* 根据 *V* 将要读取的证明输入带上的二进制位建立布尔函数 f_r（$1\le r\le|\phi|^c$）使得：$f_r(y_{r_1},...,y_{r_q})=1$ 当且仅当 *V* 根据 *r* 读取证明的第 $r_1,...,r_q$ 个二进制位 $y_{r_1},...,y_{r_q}$ 之后在工

作带上验证证明时接受该证明。显然，所有 f_r（$1\le r\le|\phi|^c$）可以在$|\phi|$的多项式时间内被构造出来。而且(1)如果 ϕ 是可满足的,则 $\wedge_{1\le r\le|\phi|^c} f_r$ 是可满足的;（2）如果 ϕ 是不可满足的,则 $f_1,...,f_{|\phi|^c}$ 中可同时满足的布尔函数至多有 $\frac{1}{2}\cdot|\phi|^c$ 个。

下面将每个布尔函数 $f_r(y_{r_1},...,y_{r_q})$（$1\le r\le|\phi|^c$）转换成 SAT 的实例 ψ_r。由于布尔函数 $f_r(y_{r_1},...,y_{r_q})$仅含 q 个布尔变量，其真值表恰有 2^q 行。将真值表中 $f_r(y_{r_1},...,y_{r_q})=1$ 对应的每个行抽取出来得到 $f_r(y_{r_1},...,y_{r_q})$的等价形式 $\vee_{1\le l_{r,k}\le|2|^q}(z_{l_{r,k},1}\wedge z_{l_{r,k},2}\wedge...\wedge z_{l_{r,k},q})$,其中 $z_{l_{r,k},j}=y_{r_j}$,如果真值表第 $l_{r,k}$ 行中 $y_{r_j}=1$；$z_{l_{r,k},j}=\neg y_{r_j}$，如果真值表第 $l_{r,k}$ 行中 $y_{r_j}=0$。最后，将析取范式 $\vee_{1\le l_{r,k}\le|2|^q}(z_{l_{r,k},1}\wedge z_{l_{r,k},2}\wedge...\wedge z_{l_{r,k},q})$改写为合取范式得到 ψ_r。注意，ψ_r 至多含有 2^q 个析取子式，每个析取子式恰有 q 个布尔变量。显然，ψ_r 与布尔函数 f_r 等价，并且由 f_r 构造 ψ_r 仅需常数时间。因此，由 $f_1,...,f_{|\phi|^c}$ 构造 $\psi_1,...,\psi_{|\phi|^c}$ 的时间复杂度为 $O(|\phi|^c)$。

最后，由 $\psi_1,...,\psi_{|\phi|^c}$ 构造 MAX-3SAT 问题的实例 ψ。这只需通过引入新的布尔变量将较长的布尔表达式转换成等价的较短的布尔表达式。例如，对于 $k>3$ 的布尔表达式 $C=w_1\vee w_2\vee...\vee w_k$ 中引入布尔变量 $u_1,...,u_{k-2}$，可以得到与 C 等价的布尔表达式

$$f=(w_1\vee w_2\vee u_1)\wedge(\neg u_1\vee w_3\vee u_2)\wedge...\wedge(\neg u_{k-2}\vee w_{k-1}\vee w_k)$$

利用上述技术将每个 ψ_r 转换成 ψ'_r 后，ψ'_r 至多含 $2^q(q\text{-}2)$个析取子式，每个析取子式恰有 3 个布尔变量。转换过程的总时间复杂度为 $O(|\phi|^c)$。令 $\psi=\wedge_{1\le r\le|\phi|^c}\psi'_r$ 得到 MAX-3SAT 问题的实例 ψ。ψ 至多含$|\phi|^c\cdot 2^q(q\text{-}2)$个析取子式，每个析取子式恰有 3 个布尔变量。

由 SAT 实例 ϕ 构造 MAX-3SAT 实例 ψ 的每个步骤均在$|\phi|$的多项式时间内完成。

如果 ϕ 是可满足的，由构造过程可知，当且仅当 $\wedge_{1\le r\le|\phi|^c} f_r$ 是可满足的，当且仅当 $\wedge_{1\le r\le|\phi|^c}\psi_r$ 是可满足的，当且仅当实例 ψ 的优化解的值为$|\phi|^c\cdot 2^q(q\text{-}2)$。

同样，如果 ϕ 不是可满足的，则 $f_1,...,f_{|\phi|^c}$ 中至少有 $\frac{1}{2}\cdot|\phi|^c$ 个函数取值为 0。由于 f_r 等价于 ψ_r 且 ψ_r 由 2^q 个析取子式构成，因此 $\psi_1\wedge...\wedge\psi_{|\phi|^c}$ 中至少有 $\frac{1}{2}\cdot|\phi|^c$ 个析取子式取值为 0。再由 ψ_r 等价于 ψ'_r 可知，$\psi=\wedge_{1\le r\le|\phi|^c}\psi'_r$ 中至少有 $\frac{1}{2}\cdot|\phi|^c$ 个析取子式取值为 0。这表明 MAX-3SAT 问题实例 ψ 的优化解不超过

$$|\phi|^c\cdot 2^q(q\text{-}2)-\frac{1}{2}\cdot|\phi|^c=(1-\varepsilon_M)\cdot|\phi|^c 2^q(q\text{-}2)$$

其中$\varepsilon_M=\dfrac{1}{2^{q+1}(q-2)}$是常数。

由此证得，SAT 问题可$(1-\varepsilon_M)^{-1}$-鸿沟归约为 MAX-3SAT 问题。 □

类似于定理 10.43，PCP 定理还可以用来建立很多其他问题的不可近似性结果。例如，集合覆盖问题和最大团问题不存在常数近似比的近似算法。

10.7.4 α,β-鸿沟归约与不可近似性

如前所述，直接利用α-鸿沟归约或者利用 PCP 定理建立α-鸿沟归约可以得到许多问题的不可近似性。借助这些的不可近似性可以进一步得到其他问题的不可近似性，这只需要将已有的α-鸿沟归约和下面讨论的α,β-鸿沟归约复合在一起即可。

定义 10.9.（**α,β-鸿沟归约**）设 $\alpha,\beta>1$ 是常数，问题 A 和问题 B 均为最小值问题（或最大值问题）且它们的实例 I_A 和 I_B 的优化解的目标函数值分别记为 $OPT(I_A)$和 $OPT(I_B)$。问题 A 可以 **α,β-鸿沟归约**到问题 B，是指存在多项式时间算法 g 和多项式时间可计算的实数值 $W(\alpha,I_A)$和 $W(\beta,I_B)$满足:

(1)f 将问题 A 的任意实例 I_A 转换成问题 B 的一个实例 I_B。

(2)$OPT(I_A)\le W(\alpha,I_A)$ 当且仅当 $OPT(I_B)\le W(\beta,I_B)$（或 $OPT(I_A)\ge W(\alpha,I_A)$ 当且仅当 $OPT(I_B)\ge W(\beta,I_B)$）；

(3) $OPT(I_A)\ge\alpha W(\alpha,I_A)$当且仅当 $OPT(I_B)\ge \beta W(\beta,I_B)$（或 $OPT(I_A)\le\alpha^{-1}W(\alpha,I_A)$当且仅当 $OPT(I_B)\le\beta^{-1}W(\beta,I_B)$）。

定理 10.44. 设 $\alpha,\beta>1$ 是常数且判定性 NP-完全问题 A 可以 α-鸿沟归约到优化问题 B,其中归约过程的多项式算法为 f，多项式可计算的实值为 $W(\alpha,I_B)$；优化问题 B 可以 α,β-鸿沟归约到优化问题 C，其中归约过程的多项式算法为 g，多项式可计算的实值为 $W(\alpha,I_B)$和 $W(\beta,I_C)$，则问题 A 可以β-鸿沟归约到问题 C。

证明. 如图 10.7 所示，对于问题 A 的任意实例 I_A，先利用算法 f 将 I_A 转换成问题 B 的实例 I_B，再利用算法 g 将 I_B 转换成问题 C 的一个实例。由于 f 和 g 均为多项式时间算法，因此问题 A 的任意实例 I_A 可以在多项式时间内转换成问题 C 的实例 I_C。

由问题 A 到问题 B 的 α-鸿沟归约可知，I_A 的解为“yes”当且仅当 $OPT(I_B)\le W(\alpha,I_B)$。由问题 B 到问题 C 的 α,β-鸿沟归约可知，后者成立当且仅当 $OPT(I_C)\le W(\beta,I_C)$。因此，I_A 的解为“yes”当且仅当 $OPT(I_C)\le W(\beta,I_C)$。

类似地，I_A 的解为“no”当且仅当 $OPT(I_C)\ge\beta\cdot W(\beta,I_C)$。 □

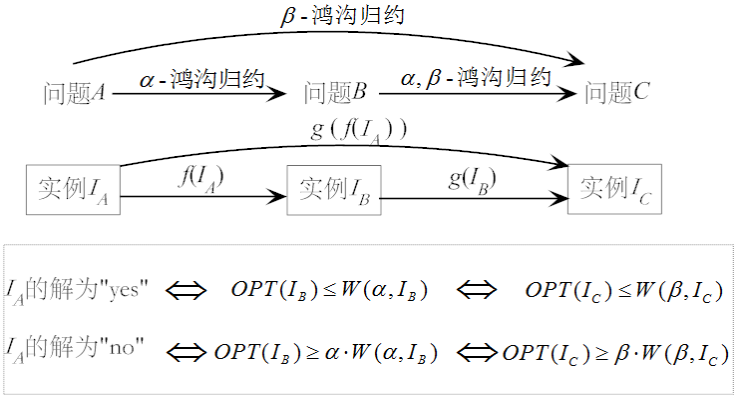


图 10.7 α-鸿沟归约与 α,β-鸿沟归约的复合

结合定理 10.38 和定理 10.44 可知，可以利用现有的不可近似性结果来建立新的不可近似性结果。这需要熟悉现有 α-鸿沟归约中的阀值，因为应用定理 10.44 将 α-鸿沟归约与 α,β-鸿沟归约进行复合时，必须确保 α-鸿沟归约中的阀值和 α,β-鸿沟归约中的一个阀值相同。

习题

10.1利用第 7 章图的匹配算法，给出定点覆盖问题的一个近似比为 2 的近似算法。

10.2将 10.2.3 节最短并行算法中任务的任意顺序修改为处理时间的递增顺序，证明修改后的算法的近似比为 4/3。

10.3利用最小生成树算法和最大权值匹配算法（参见第 7 章）给出满足三角不等式的旅行商问题的一个 3/2-近似算法。（提示：利用最大匹配改造最小生成树使其存在哈密顿环，分析近似比时注意使用三角不等式）。

10.4Steiner tree 问题的输入是加权连通图 $G=(V,E)$和一个顶点子集 $T\subseteq V$，输出是总权值最小的一个边集子集$E'\subseteq E$ 使得在 T中的任意两个顶点在 $G'=(V,E')$上是连通的。试利用最短路径算法和最小生成树算法为 Steiner tree 问题设计一个近似比为 2 的近似算法。（提示：分析近似比时需要利用 10.2.4 节的结论）

10.5试修改 10.3.1 的集合覆盖算法求解加权集合覆盖问题，并分析它的近似比。

10.6最小多路割问题的输入是连通加权图 $G=(V,E)$和顶点子集 $T\subseteq V$，其中$|T|>1$ 每条边 $e\in E$ 的权值记为 $w(e)$，输出总权值最小的边子集 $E'\subseteq V$ 使得$\forall u,v\in T$ 在 $G=(V,E\setminus E')$中不连通。试利用最小割（最大流）算法，给出最小多路割问题的一个基于贪心思想的近似比为 2 的近似算法。（提示：如下建立近似解和优化解之间的联系，近似解和优化解均会将$\forall v\in T$ 与 $T\setminus\{v\}$分割开）。

10.7考虑 10.6.3 节随机舍入算法的如下改进。调用 SetCoverRRounding 算法 $b\log n$ 次，其中 b 满足 $e^{-b\log n}\le (4n)^{-1}$，将各次调用输出的集族求并集得到 C'，将 C'作为最终的输出。证明：

(1). X 未被 C'覆盖的概率不超过 1/4；

(2). $E(w(C'))=OPT_f\cdot\log n$；

(3).C'覆盖 X 且 $w(C')\le OPT_f\cdot 4b\cdot\log n$ 的概率不小于 1/2。

10.8加权顶点覆盖问题的输入是图 $G=(V,E)$，其中每个顶点 $v\in V$ 的权值为 $w(v)$,输出是总权值最小的顶点子集 $C\subseteq V$ 使得 E 中每条边至少有一个端点位于 C 中。

(1).将加权顶点覆盖问题表示成整数规划；

(2).用舍入法设计加权顶点覆盖问题的一个近似算法并分析其近似比。

10.9利用线性规划方法，设计一个 2-近似算法求解如下的 SONET 电话负载问题。

输入：含有 n 个顶点的环（其中所有顶点按顺时针方向列出得到 0,1, 2,…, n-1）。一个电话呼叫集合 C，其中$(i, j)\in C$ 表示节点 i 呼叫节点 j。任意电话的路由既可以按顺时针方向进行也可以按逆时针方向进行。对于 C 中电话的路由策略，边$(i, i+1 \bmod n)$的负载为通过该边的电话个数 L_i。环的总负载为 $\max_{1\le i\le n} L_i$。

输出：C 中电话的一个路由策略，使得环的总负载最小。

10.10　非二分图上的加权匹配问题的输入是图 $G=(V,E)$，其中每条边 $(i,j)\in E$ 具有非负权值 w_{ij}，输出是总权值最大的匹配 M。

考虑如下的贪心算法。维护一个边集子集 M,其初始化为空集。重复地将权值最大且与 M 中的边无公共端点的边添加到 M 中,直到 M 中无法继续添加边为止。记 z 是 M 中所有边的权值之和，z^*是问题的优化解的代价。根据下面的提示，证明上述贪心算法的近似比为 2。

(1).证明下面的线性规划的目标函数最优取值z_{LP} 是z^*的上界。

$$min \quad \Sigma_{x_i\in V} x_i$$

$$\text{s.t.} \quad x_i + x_j \ge w_{ij} \quad \forall (i, j)\in E$$

$$x_i \geq 0 \qquad \forall i \in V$$

(2).利用贪心算法得到的匹配M构造上述线性规划的一个可行解x,证明该可行解的代价至多为$2z$。据此证明 $2z \geq z^*$。

10.11 证明:10.2.3 节讨论的最短平行调度问题不存在 3/2-近似算法。（提示：采用类似于装箱问题的 3/2-鸿沟归约）

10.12 证明：如果 10.4.2 节讨论的设施定位问题的距离函数是任意的非负加权函数，则设施定位问题不存在常数近似比的近似算法，除非 $NP=P$。（提示，将集合覆盖问题问题归约到修改后的设施定位问题）