第2卷 第2期
1983年12月

运 筹 学 杂 志
CHINESE JOURNAL OF OPERATIONS RESEARCH

Vol. 2 No. 2
Dec., 1983

组合时间表理论(二)

R. L. Graham 著　　高 济 译

求特殊问题的最优时间表

要寻找一个有效方法来构造最优时间表，一般说来似乎永远做不到，但在时间表问题中，却存在几个使人感兴趣的特殊情况，使得求最优时间表是可能的．在本节中，我们要来讨论其中两种情况，同时为了有助于理解为什么这些途径能行得通，我们要简单谈一谈隐藏在它们后面的基本原理．

在时间表问题中，其复杂性大部分来自于先后限制以及加工时间的复杂的数论性质可能出现的复杂结构．在我们的第一个特例中，我们将把这两个因素加以限制：假定所有的加工时间皆为1，先后限制则是树状的．这就是说：每一项工作 T 至多只有一件后继者，也就是至多只有一个箭头从 T 引出．图10中就是树状先后限制的一个例子．

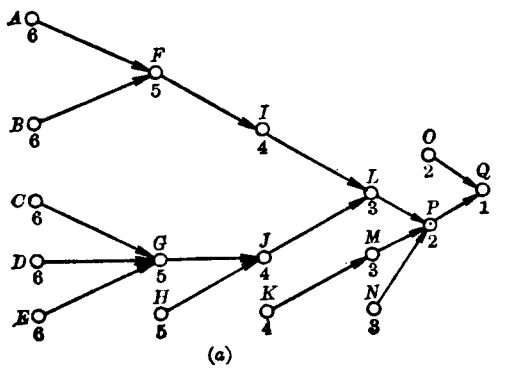


图10 对于等长而又以树状先后限制的工作(a)来说，CP时间表是最优的(b)

在这些特殊限制之下，关键路时间表总是最优的，这一由加州大学胡德强在1961年证明的结果是这一领域中的第一个结果．对于这种情况，要构造出CP时间表我们只须对于每一项工作 T，标上以 T 打头的最长的链的长度 $L(T)$．这一数目 $L(T)$ 也称为 T 的水平(在图1中我们已对每一工作标上水平)．只要有一台空出来，我们总是选取具有最长水平的工作来加工．图1中也表示出图中所显示的工作关于三台机器的一张时间表．

第二种我们所要考察的情况将允许先后限制可以任意(不一定是树状)，但我们仍然要求所有的加工时间为1．而且，我们只考虑两台机器．对于这种情况，现今已知有三种基本上不同的方法来构造最优时间表，每种方法多少都有点复杂，正如下面两台机器的算法这一节所说明的那样，通过这一节，我们希望读者对于那些可以成功地应用于特殊的时间表问题的种种技巧得到一些概念．人们可能会盼望：将这些想法加以扩充，对于某些相关的问题，例如加工时间为1的三台机器的问题，会得到同样成功的算法．但在目前，这一令人垂涎的问题仍然完全没有得到解答．应该指出的是：一个"稍微"广一些的问题，即加工时间为1或2的两台机器的问题，最近已被证明为

本文1981年1月23日收到．

一 NP 完备问题，正如我们前面所说的，这就提供了强烈的理论证明：在这种情况，不存在可以保证构造出最优时间表的有效算法.

两台机器的算法

对于加工时间等长、两台机器的情况，现在已经有许多方法可以用来定出一个最优时间表，下面是其中的三个：

FKN 算法(M. Fujii, T. Kasami, K. Ninomiya(1969))　这一方法的主导思想是：仅当两件工作 A 和 B 不能相比较时，也就是说，其中一个并不需要等另一个完工才能开始时，它们才能在同一时间内加工. 我们把工作中任何两个不能相比较的工作用线连起来，这样就得到一个图形(称为所给的这组工作的不可比较图). 然后在图中求出连接两两工作之间的最大数目的线段组，使得其中没有两条线段通过同一工作. 在本例中，线段 A-B, C-D, E-F, G-I, H-J 就是这样的一组线段. 这些"线段对"表示可以同时加工的工作，其它的则在任一时刻只能加工其中一个. (为了真正产生一个有效的时间表，偶而我们还需要作数量不大的互换. 比如说，假若四件工作 P、Q、R、S 的先后限制为 $P\to Q$、$R\to S$，则 FKN 算法可以挑选 P-S 与 R-Q 来同时加工，而这是不可能的. 但我们可将 R 与 S 相交换，加工 P-R 及 S-Q 而不会引起麻烦.)因为对于具有 n 个点的图，要求出其中不相连接的最大数目的线段组，这已有一般性的方法：至多需要 $n^{5/2}$ 步即可求出，故 FKN 法确实满足我们所说的关于有效性和最优性的要求.

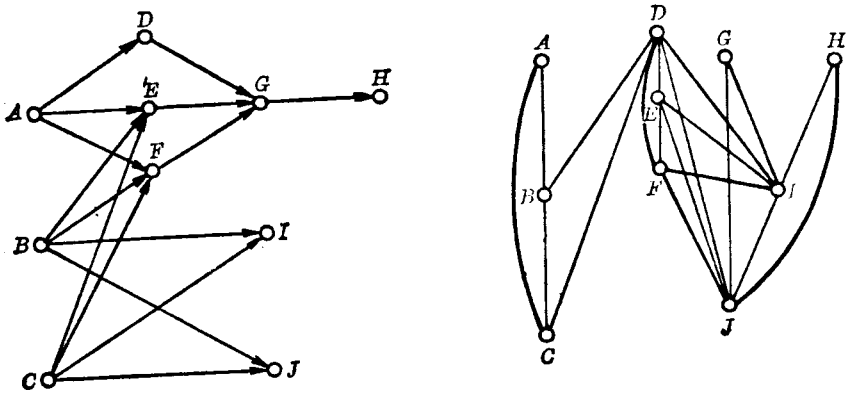


图 10.1　不可比较图

CG 法(E. Coffman 与 R.Graham(1972))　这条途径的主要精神乃是前面所讨论的水平算法. 其主要思想是：对每件工作赋予一些数字，这些数字则是根据该工作所有的后继者已经被赋予的数字来赋予的，而不是象水平算法那样只是根据单独一个后继者来赋予. 为了应用这一算法，我们必须先将所有多余的先后限制去掉，换言之，若 $A\to B$, $B\to C$，则不必包括 $A\to C$，因为这是自然成立的[除去这些多余的边这一件事，我们称之为作先后限制组的"传递归并"(transitive reduction)]，对于包含 n 件工作的组，这至多需要 $n^{2.81}$ 次运算即可完成). CG 法是：先将一件无后继的工作赋予数字 1，然后对于每一件其所有后继皆已赋予数字的工作，取其后继者的数字作一单降序列，然后挑选依所谓"字典"序具有最小序列

的工作（例如(5, 3, 2)依“字典”序就在(6, 1), (5, 4, 2), (5, 3, 2, 1)之前），依次标上数字. 在所有的工作依次用数字1到n标上之后，我们就从具有数字n的工作开始按数字的单降次序将各工作进行加工，这样就作成了所要的时间表.

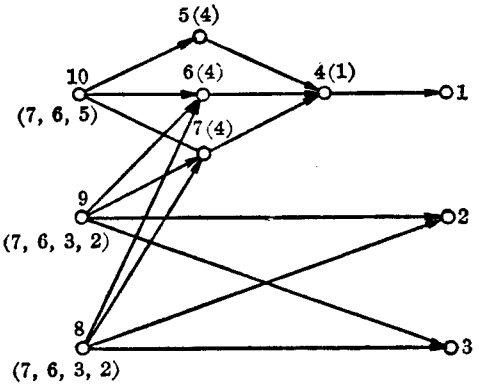


图 10.2

图10.2所示就是CG标号的一个例子. 注意：在D、E、F、G、H、I、J已经标号之后，在A、B、C三者之中，我们先挑选B或C来标号，因为A的后继序列(7, 6, 5)按字典序比B和B的后继序列(7,6,3,2)来得大. 其结果，A得到标号10，因而先被加工. CG算法基本上所做的，就是将后面跟上一长串工作或者说具有多个后继工作的那种工作标上较大的数字，因而在排时间表时，就倾向于把它们放在前面. 注意：假若我们仅使用简单的CP时间表，即将后继者最多(或最长)的那种工作先加工，那就没有理由不将B或C先加工，其结果就得到了下面的时间表：

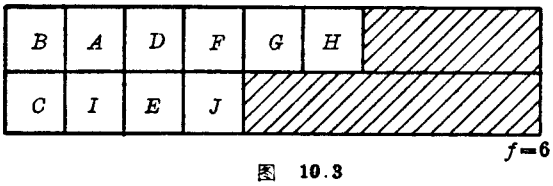


图 10.3

GJ 算法(M. R. Garey, D. S. Johnson(1975)) 在施行这一算法时，我们须将先后限制所蕴含的所有箭头画上(这称为作成这一组先后限制的“传递包”，正如同传递归并一样，对于n件工作的组来说，这至多需要$n^{2.81}$次运算即可完成. 假设我们要设法在某一截止期d之前将所有的工作做完，GJ算法就是要计算某些不得不保证的截止期(它们总是$\leqslant d$)，假如所有的工作真能在时间d之前完成的话. 其作法是：先把所有那些没有后继的工作都标上截止期d，于是对于那些工作：它的后继都已标上截止期的，我们采取如下作法：对于T的每一后继所标上的截止期d'，计算出具有小于或等于截止期d'的那些后继的个数$N_{d'}$，作$d'-\left[\frac{1}{2}N_{d'}\right]$，对于$T$的每一后继有一$d'-\left[\frac{1}{2}N_{d'}\right]$. 我们将其中的最小者规定为$T$的截止期(式中$[x]$表示大于或等于$x$的最小整数).

容易看出，若T具有$N_{d'}$个截止期小于或等于d'的后继，要所有的截止期都必须得到保证时，则T必须在时刻$d'-\left[\frac{1}{2}N_{d'}\right]$之前完成. 在所有的截止期皆已标上之后，要作GJ时间表，我们只须依截止期早晚的次序来按排工作的先后即可. Garry和Johnson所证明的定理是：若有一个能在时刻d之前完成全部工作的时间表，则用此法即可得出一个. 由此可

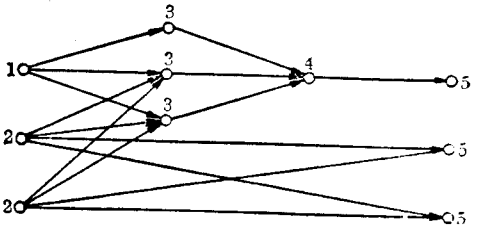


图 10.4

知,GJ 时间表具有最小的完工时间(在造表时我们从什么样的 d 出发,这一点并不重要). 上图中我们是使用 $d=5$ 来对工作标时间表. 显而易见,最早的截止期(1)属于 A,我们已经看出,无论是在哪张最优时间表中,皆必须先将 A 加工.

下面是三种算法中任一种所产生的最优时间表:

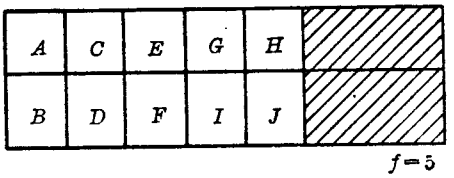


图 10.5

装 箱 问 题

在最令人感兴趣的种种类型的时间表问题中,有一种就是把通常的问题倒过来提,现在不是固定了机器的台数,设法尽快把工作完成,而是问: 要把所有的工作在所给的截止期 d 之前完成, 至少需要多少台机器(我们暂时只限于讨论无先后限制的情形). 这一新的时间表问题,通常称为"装箱问题". 这一问题的标准提法如下: 给了一组项目 I_1, $I_2\cdots$,每一项目 I_k有一重量 w_k. 另外还有一些同样性质的容器,称作"箱", 每个箱能容纳的重量至多为某一给定的数 W,称为"箱的容量", 我们的目的就是要用尽可能少的箱把所有的项目都装进去(用时间表的语言来说,箱就是机器, 我们的目的就是要用尽可能少的机器将所有的工作做完,但是对每一台机器所给的工作皆不能超过该机器的容量,即截止期, 也就是所有的工作必须在这之前完成的期限).

装箱问题出现于许多的实际情况中, 而且以种种不同的面貌出现,例如: 要从一些标准长度的管子上剪下某些所需要的长度的管子,白铁工人就需要设法用尽量少的(标准长度的)管子来达到这一目的; 裁纸工人必须从具有标准宽度的纸卷裁下具有各种宽度和不同数量的纸卷来供给顾客的需要,他需要用尽可能少的(标准)纸卷来达到这一目的; 电视网络需要将它的具有不同长度的商业广告编排到尽可能少的标准长度的(电台)间歇中去. 另外一个容易想象的例子是: 一个人一手拿着不同种类的许多信, 另一手拿着一把二角五分的镍币, 走到一台标准邮政(自动)售票机前, 他将考虑如何使用手里的镍币.

一般说来,装箱问题可能非常困难. 对于编排最优装箱表(即使用最少的箱子),目前所知道的方法都包含着: 将所有可能的装法加以考察,然后选出其中最好的一个. 对于非常小的问题来说,这也许是可行的,但当问题的规模变得适当大时, 这些方法便无法使用. 我们看看图 11 中的问题就知道. 通常用来克服这种困难的途径就是要设计出有效的启发式方法,它们也许并不总是产生最好的装法, 但却能保证找到相当接近于最优装法的装法, 在前面提到的时间表问题中我们已经看到具有这种观点的例子. 算法理论中有一些非常深刻的结果, 就是联系着研究装箱问题的这条途径而得到的. 我们现在来谈谈其中几个结果.

1415926535	5820974944	8979323846	5923078164	2643383279
8214808651	4811174502	3282306647	8410270193	0938446095
4428810975	4564856692	6659334461	3460348610	2847564823
7245870066	7892590360	0631558817	0113305305	4881520920
3305727036	0744623799	5759591953	6274956735	0921861173
9833673362	6094370277	4406566430	0539217176	8602139494
0005681271	1468440901	4526356082	2249534301	7785771342
4201995611	5187072113	2129021960	4999999837	8640344181
5024459455	7101000313	3469083026	7838752886	4252230825
5982534904	8903894223	2875546873	2858849455	1159562863
0628620899	5028841971	8628034825	6939937510	3421170679
8521105559	5058223172	6446229489	5359408128	5493038196
4543266482	3786783165	1339360726	2712019091	0249141273
4882046652	9628292540	1384146951	9171536436	9415116094
1885752724	8193261179	8912279381	3105118548	8301194912
2931767523	6395224737	8467481846	1907021798	7669405132
4654958537	7577896091	1050792279	7363717872	6892589235
2978049951	5981362977	0597317328	4771309960	1609631859
5875332083	3344685035	8142061717	2619311881	7669147303
9550031194	8823537875	6252505467	9375195778	4157424218

图 11

图 11 中的 100 个重量能装入 10 个容量为 15×10^{10} 的容器中吗？即使使用当前世界上所有的计算机的力量，看来也没有希望回答这一问题．这一例子，虽然(在当前)并不特别现实，但却可以说明即使在解决相当小型的装箱问题时也可能遇到的巨大困难．

开始时，我们先把安装的项目的重量随意排成一张表 $L=(w_1, w_2, \cdots)$，我们就用重量来表示项目，这不会引起混淆．有一个显然的装放重量为 L 的项目的装法，称为 L 的适合先装法，即按项目在 L 中的次序装放，先将一个箱装到不能再装时(即要超过限额时)再装下一个箱．说得确切些，当要轮到装 w_k 时，是把 w_k 装在能够容得下 w_k 的那种箱 B_i 中具有最小指标 i 者，因而每一 B_i 中的项目的重量不超过 W(当然，一开始，我们就假定所有 w_k 的重量皆不超过 W，否则装箱就不能进行)．

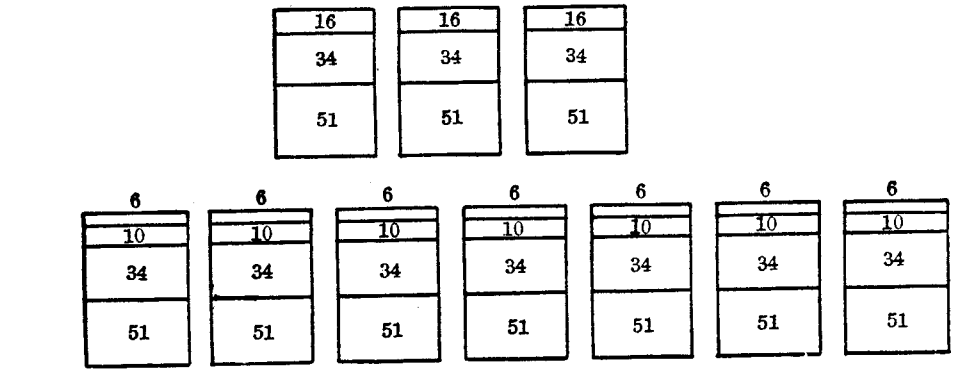
我们用 $FF(L)$ 来表示采用适合先装法于表 L 时所需要的箱子数；用 $OPT(L)$ 表示采用最优装箱法于 L 时所需的箱子数．关于成效保证的一个自然问题就是：$FF(L)$ 究竟能比 $OPT(L)$ 大多少？J. Ullman 所给的答复是非常有趣的．Ullman 在 1973 年证明：对于任何重量表 L，常有

$$FF(L)\leqslant\frac{17}{10}OPT(L)+2.$$

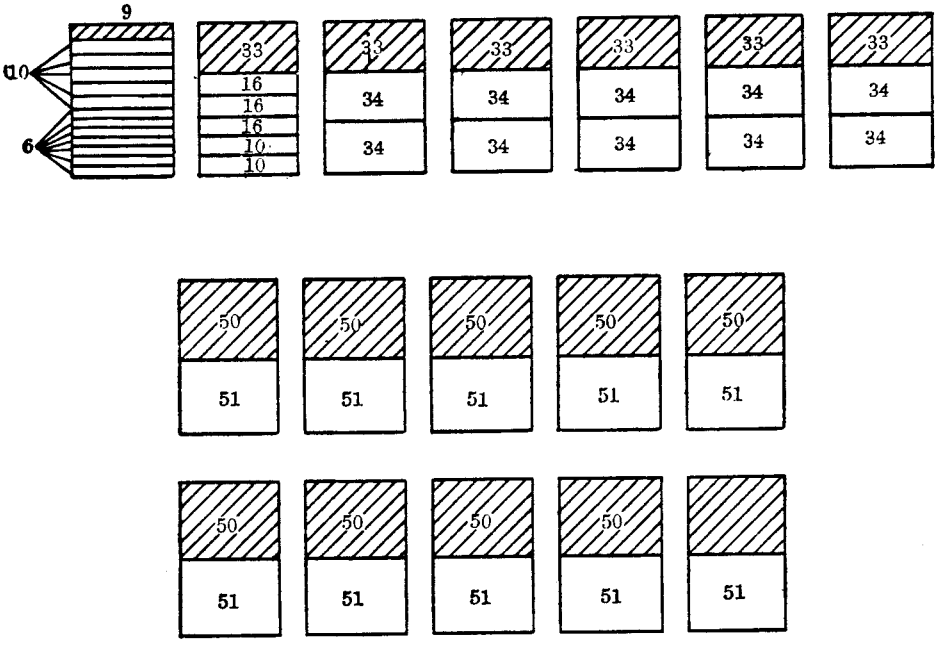
他又证明了“没有预料到的分数 $\frac{17}{10}$ 是不能用更小的数字去代替的”．在图 12 中，我们给出了一个例子，其 $FF(L)=17$，$OPT(L)=10$，因而 $FF(L)=\frac{17}{10}OPT(L)$，人们猜想 Ullman 的不等式在去掉 2 之后恒成立．但现在还不知道当 $OPT(L)$ 变得非常大时，在此种情况等号是否成立．

6	6	6	6	6	6	6
10	10	10	10	10	10	10
16	16	16	34	34	34	34
34	34	34	34	34	34	51
51	51	51	51	51	51	51
51						

重　量　表



$W=101$ 的 10 个箱的最优装放法



$W=101$ 的 17 个箱的适合先装法

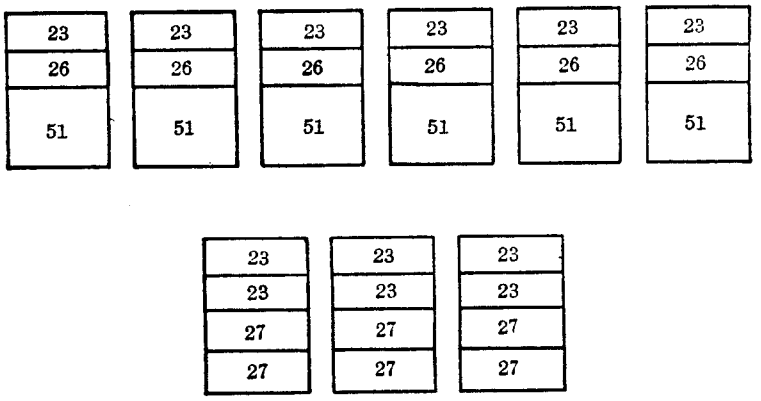
图 12　所给的重量可以最优地装在 10 个容量为 101 的箱中. 但是适合先装法——将诸项目按表中出现的次序装放——则需要 17 个同样容量的箱子. $\frac{17}{10}$ 是在适合先装法之下所出现的最坏的数.

相当奇怪的数字 $\frac{17}{10}$ 的出现，显示出最终与所谓的“埃及”分数，也就是分子为1的分数有关．这种分数在一本已知的最早的数学手稿，即 Aahmes(公元前1690年)的著名 Rhind 抄本中曾作了广泛的研究．在当时通常都将分数量表示成“埃及”分数，或单位分数之和．例如将 $\frac{25}{28}$ 写成 $\frac{1}{2}+\frac{1}{4}+\frac{1}{7}$．

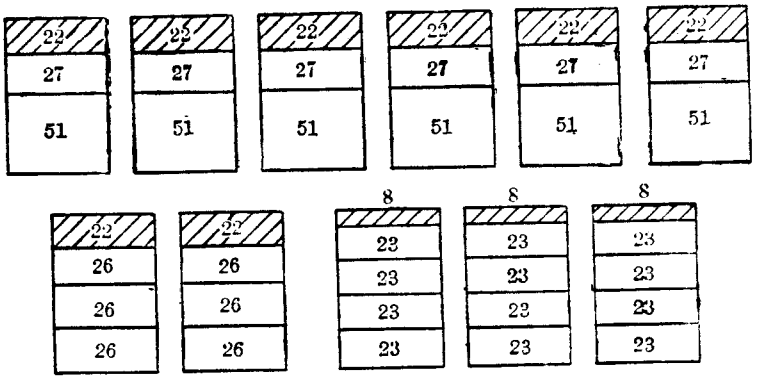
图12所给出的例子中的 $FF(L)$ 之所以如此大的原因是由于把重量大的项目皆放在最后来装．正如编制相互独立的工作的时间表那样，把重量较大的项目放在表的前头，而把重

51	27	26	23	23
51	27	26	23	23
51	27	26	23	23
51	27	26	23	23
51	27	26	23	23
51	27	26	23	23

重 量 表



关于容量 $W=100$ 的箱子的最优装箱



递减适应先装法需要11只箱子

图13 设箱子容量 $W=100$．若重量表依单降次序排列，则适合先装法需要11只箱子，最优法为9．将各重量重复 n 次，则可构造出一个最大的例子，它具有比数 $\frac{11}{9}$．

量较小的项目放在末了来填写，这样做就会更明智一些．说得更明确一些，我们把 L 的重量按从重到轻的顺序排成一张新的下降表：重量大的总是排在重量小的前面(在前面编制无先后限制的 CP 时间表时就是这样做的)，然后将适合先装法应用于此新表．这种装放法称为 L 的递降适合先装法(first-fit decreasing)；此种装法所要的箱子数记作 $FFD(L)$．

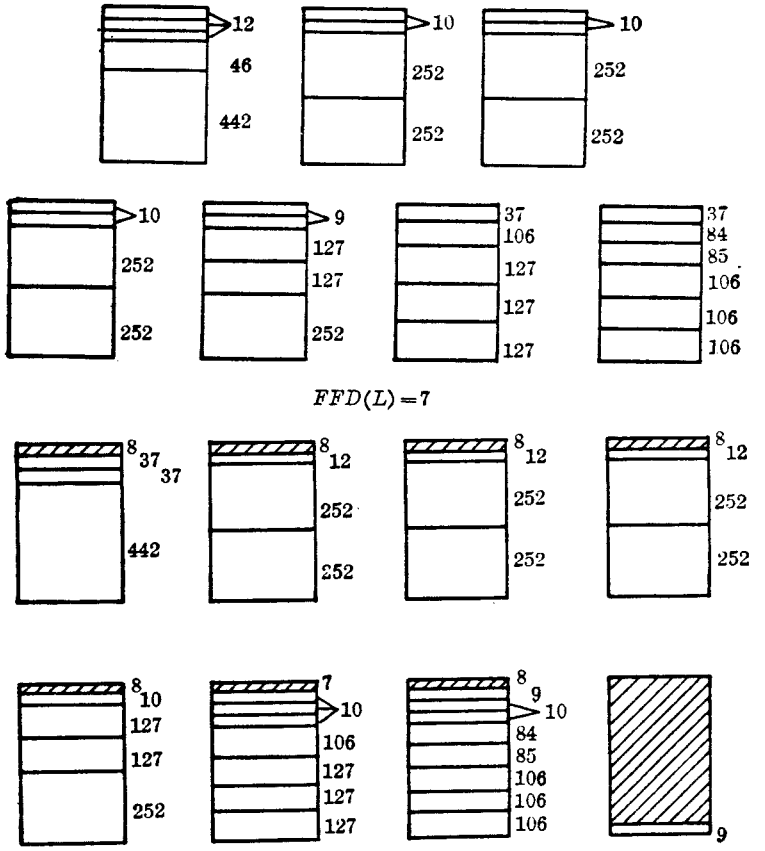
递降适合先装法的效果却是出奇的好：对于任何的重量表 L，总有

$$FFD(L)\leqslant\frac{11}{9}OPT(L)+4.$$

于是，对于那些要求大量数目的箱子的重量表来说(此时上式中的 4 相对来说无关重要)，递降适合先装法可以保证：比起理论上的最优装箱法来说，不会多用大致 22% 的箱子，而在使

442	252	127	106	37	10	10
252	252	127	106	37	10	9
252	252	127	85	12	10	9
252	127	106	84	12	10	
252	127	106	46	12	10	

重　量　表



$FFD(L\downarrow)=8$ 于此 $L\downarrow=L-\{47\}$．

图 14　表中所列的重量在递减适合先装时，需要 7 个容量为 524 的箱子，但若将一个重量从表中除去，则同一算法需要 8 只箱子．

用适合先装法时，若遇到的重量表特别差，则就可能出现70% 的浪费. 图 13 中所给的例子说明，$\frac{11}{9}$ 这个系数不可能用任何更小的数目代替.

$FFD(L)$的这一上界$\left(\text{即 } \frac{11}{9}OPT(L)+4\right)$样子很简单，但这丝毫没有表明人们在证明这一式子时所遇到的本质性困难. 目前所知的唯一证明是 David Johnson 所给的，其长达 75 页. 与 $\frac{17}{10}$ 的情况相反，当前还没有人知道分数 $\frac{11}{9}$ 的出现是如何来的.

各种装箱算法显得都很复杂，其原因之一就是存在着某些与直观相反的异常现象. 例如假若我们所要装放的重量表为

442	252	127	106	37	10	10
252	252	127	106	37	10	9
252	252	127	85	12	10	9
252	127	106	84	12	10	
252	127	106	46	12	10	

又假定箱子容量 w 为 524. 在图 14 中，我们给出一个递降适合先装者；它需要 7 个箱子. 但若将重量 47 从表中除去，则对此缩小了的表 L'，递降适合先装法需要八只箱子(参看图 14). 毫不奇怪，在直截了当的去研究装配问题时，这种现象可能引起严重的困难.

图 案 安 排

刚才所说的装箱问题的一个自然推广就是所谓的二维装箱问题. 在这问题中，所给的是一些大小和形状不同的平面区域(图案)，要求把它们放置在尽量少的"标准"区域内，但不能有相互迭置处. 常见的例子有：将裁衣纸样摆放在一些衣料上；将鞋帮各部分的纸样摆放在一些不规则的皮张上；等等. 可能有人认为：在这里，由于牵涉到的部件形状不规则而造成困难. 但是，即使对于非常有规则的部件，最优安放应如何，也常常远不是一目了然的. 实际上，这类问题也许永远得不到完全解决.

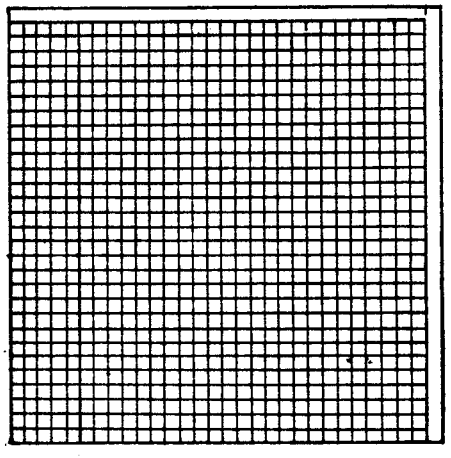


图 15 将 $(100,000)^2$ 个单位平方放置到一个边长为 100,000.1 的大正方形内，留下 20,000 个多一点的平方单位没有盖上，但没有一个单位正方形可以放入未盖上的区域.

作为一个例子，我们现在来考虑下面一个简单的几何安放问题：在一个边长为 s 的巨大正方形内能够放置多少个互无迭置的单位正方形？ 当然，假若 s 等于某一整数 N，则易看出：正确的答案是 N^2. 但若 s 不是整数，比如说，$s=N+\frac{1}{10}$. 在这种情形，我们该怎么办？ 当然，一种可行的做法就是用 N^2 个单位正方形以显然的方式(参见图 15)去填入一个 $N\times N$ 的子正方形，然后把那些没有盖

上的部分作为不可避免的浪费而放弃$\left(\text{将近}\frac{s}{5}\text{个平方单位}\right)$. 但这真是我们所能采取的最好办法吗？令人惊异的是：答案是否定的！最近 Paul Erdös、Hugh Montgomry 以及本文作者证明：当 s 甚大时，对于任何 $s\times s$ 正方形，真正存在安置方法，使得留下没有盖住的面积至多为 $s^{(3-\sqrt{3})/2}\approx s^{0.634\cdots}$ 个平方单位；当 s 甚大时，这显然比用简单安置法留下没有盖住的 $\frac{s}{5}$ 平方单位要小得多. 比如说，当 $s=100,000.1$ 时，习惯的安置法(如图 15)放入 $100,000^2=10^{10}$ 个单位正方形，留下 $20,000(=100,000/5)$ 多一点平方单位没有盖上. 但因 $(100,000.1)^{0.634}=1479.11$，因此通过精明的安置，就可以将 6000 个以上额外的单位正方形放进去. 对于充分大的 s，数字 $s^{0.634}$ 可能不是最好的界限；要确定不可避免地不能盖上的区域的确切的阶(order)是什么，这是非常困难的，虽然 $\sqrt{s}=s^{0.5}$ 看来很象是所要的结果.

本节中我们所考虑的这一特殊时间表模型虽很简单和基本，但仍具有足够的结构来显示出一些更为复杂和更为现实的情况及预料不到的典型性质. 出现的事情是：机器对某些相对来说不甚重要的工作开始进行处理(因为不允许有不必要的停顿)，在已经开始之后，直到这些工作完成才能停止，因而将一些当时变得更为紧急的工作拖延下来，在现实世界中，排时间表时，这种作法并不是少见的. 这一事件说明我们的模型假设是有道理的.

如何确定一个次序，使得当工作遵循这一次序被进行处理时可以使总的完工时间达到最小？这是一个极为困难的问题. 在大多数现实复杂情况中，我们不得不满足于得到某些(我们希望的)相当接近于最优解的答案. 幸而我们常常可以找到一些有效算法来求近似解；在处理时间表问题时，这常常是最有用的途径，事实上，对于 NP-完备问题，在目前，这是唯一的提供某些希望的一般性方法.

当然，我们的基本模型可以有许多推广. 例如，它们可以包括工作完成之前可以有所中断；对于工作，可以引进各种资源上的要求；工作可以随机到达；机器相互不一样；测量效果的尺度不同等等. 将这些推广了的模型，用我们上述的分析方法来处理时，今天的研究工作者很快就能看出那些非常困难的时间表问题.

(上接第 60 页)

定理 4　设 Q^{1k}、Q^{2k} 满足 (F)，$\bar{x}$ 是迭代点列的一个极限点. 若存在一个子列 $\{x^k,\ k\in K\}\to\bar{x}$，$J_k=J$，$J_0(x^k)=J_0$，$\forall k\in K$，$J_0(\bar{x})\subset J$，$\overline{\lim_{k\in K}}\nabla f(x^k)'d^k<0$，$\overline{\lim_{k\in K}}(\bar{\alpha}_k-\alpha_k)=0$，且对 $\forall i\in J_0(\bar{x})\bigcap(J\backslash J_0)$，有 $\overline{\lim_{k\in K}}\sum_{j=1}^{n}q_{ij}^{2k}\nabla_j f(x^k)/(b_i-a_i'x^k)<+\infty$，

则 $\bar{x}$ 是一个 K-T 点.

我们可验证[1]、[2]、[3]及其他一些算法满足定理 3 或定理 4 中的条件，因而具有全局收敛性. 我们可给出一个简单的转轴算法，用以产生 J_k，使得满足条件 $J_0(\bar{x})\subset J$.

参 考 文 献

[1] Ritter, K.:"Convergence and superlinear convergence of algorithms for linearly constrained minimization problems", in Nonlinear Optimization, Theory and Algorithm, L. C. W. Dixon, E. Spedicato and G. P. Szego", eds., 1980, 221～252.

[2] Zangwill, W. I.; "The convex simplex method", Management Science, 14(1967),221～238.

[3] 王长钰:《非线性规划的一个可行方向法》,数学学报,25:1(1982), 15～19.