DOI:10.13976/j.cnki.xk.1991.s1.015

0-1背包问题的近似算法

郑中旺

(中科院沈阳自动化研究所)

摘要 本文将给出0-1背包(Knapsack)问题的几个近似算法，它们都是对Greedy算法的改进. 对100个例子进行了计算和分析，结果令人满意.

关键词: 计算机科学，组合最优化，计算复杂性，NP-Complete问题

1 引言

设 $\{a[1],\ \ldots,\ a[n];\ v\}$ 是自然数，是否存在 $\{x[1],\ \ldots,\ x[n]:\ 0 \text{ or } 1\}$ 使得

$$a[1]*x[1]+\ldots+a[n]*x[n]=v \tag{1}$$

这是0-1背包问题的判定形式(Decision form)，它的最优化形式(Optimization form)是

$$\max a[1]*x[1]+\ldots+a[n]*x[n]$$
$$a[1]*x[1]+\ldots+a[n]*x[n]\leqslant v \tag{2}$$

背包问题尽管形式上非常简单，但计算起来却非常困难. 从计算复杂性上看，背包问题属于NP-Complete问题. 到目前为止，这类问题都没有好的算法，即多项式时间的算法. 用于精确求解背包问题的算法主要有分支定界和动态规划，但它们都是非多项式时间的算法. 当问题的规模较大时，实际问题往往是这样，运行时间就会激增，这些算法就变得完全不能用，因此寻求近似算法是非常必要的. NP-Complete问题作为计算复杂性中的核心问题，越来越受到人们的重视. 研究人员试图找出多项式时间算法的努力都没有成功，而试图否定它的努力也没有结果. 目前人们普遍认为NP-Complete问题没有多项式时间算法，从直观上看情况确实如此，但缺乏有力的证据. 科学发展的历史告诉我们，直观有时也靠不住，有时也会出问题. 笔者认为，要给出NP-Complete问题的否定解答，是根本不可能的，无论从逻辑上讲还是从方法论上讲都是如此. 背包问题出现于装货、下料、财政预算等方面，有些排序、调度方面的问题，本质上也归结为背包问题. 例如机床负荷问题：假设 n 个工件 $\{p[1],\ p[2],\ \ldots,\ p[n]\}$ 都可以在机床 M 上加工，工件 $p[i]$ 的加工时间为 $a[i]$，问题是在时间 T 内使得机床 M 的负荷尽可能大，这就归结为背包问题的最优化形式(2)；如果问题是在时间 T 内能否使机床 M 的负荷达到100%，这就归结为背包问题的判定形式(1). 背包问题是组合最优化中的典型问题. 在目前已知的NP-Complete问题中，背包问题的形式最简单，而且它与数论、组合论等其它基础学科的联系也最明显. 研究背包问题的近似算法，不仅是出于生产过程的实际需要，而且在研究过程中，可能会发现一些问题的内在规律、固有属性，这对于推进NP-Complete问题的研究，具有真正的帮助.

2 算法

为了准确地叙述这些算法，我们将用程序设计的方式给出，并用Turbo Pascal 5.5编

收稿日期：1991—10—25. 本文得到国家高技术计划CIMS主题的资助.

写. 每个算法都以判定和最优化两种形式给出，虽然这两种形式有着相同的时间复杂性，但在实际计算时，判定形式通常要比最优化形式快. 背包问题的数据由随机数产生. 为了对近似算法进行有效的检验，我们假定所有要计算的问题都有解，而且 v 就是问题的最优解. 否则就无法进行这样的检验，因为当问题的规模较大时，要求出最优解是非常困难的. 下面的程序用来产生数据 $\{a[1], \ldots, a[n]\}$ & v，并且按从大到小的顺序加以排列.

```
procedure CREATEDATA;
  var
    i, k, v: longint; a: array[1. n] of integer;
  begin
    randomize;
    for i:= 1 to n do a[i]: = random(maxint);
    repeat
      k:= 0;
      for i:= 1 to n-1 do if a[i] < a[i+ 1] then
      begin
        k:= a[i]; a[i]:= a[i+ 1]; a[i+ 1]:= k; k'= 1
      end;
    until k= 0;
    v:= 0;
    for i= 1 to n do v:= v + a[i] * random(2)
  end;
```

2.1 标准的 Greedy 算法

Greedy 算法是组合最优化中的基本算法之一，它的程序结构非常简单，计算速度也非常快，是我们进行程序设计的基本出发点，有些算法就是从它演变而来的，E.Horowitz 和 S.Sahni 的书中就有一章专门用来讲它. GREEDY 算法的计算复杂性为 O(n).

过程说明: 所求问题的解是 s.

```
  procedure GREEDY;
  var
    i:integer; s:longint;
  begin
    s:= 0;
    for i:= 1 to n do if a[i] < vthen
    begin
      s:= s + a[i]; v:= v - a[i]
    end
  end;
```

2.2 改进的 Greedy 算法 A

s 的定义和上述标准的 GREEDY 算法中的 s 相同，该算法的时间复杂性为 O(n * n).

判定形式：这个算法的计算过程是这样的，首先对{a[1], …, a[n]; v}用标准GREEDY算法求解，如果 $s=v$，那么停机；否则对{a[2], …, a[n]; v}用标准GREEDY求解，如果 $s=v$，那么停机；否则对{a[3], …, a[n]; v}用标准GREEDY算法求解，直到 $a[i]+\ldots+a[n]<v$. 过程说明：如果 $t=$true那么问题有解；如果 $t=$false那么问题无解.

```
procedure GREEDY__A; { Decision form }
var
    i, k:integer; s, u :longint; t: boolean;
begin
    k:= 0;
    repeat
        k:=k+1; s:= 0; u:=v;
        for i:=k to n do if a[i] ≤ u then
        begin
            s:=s+a[i]; u:=u-a[i]
        end;
    until (s = v) or (k = n);
    t:= (s = v)
end;
```

最优化形式：首先对{a[1], …, a[n]; v}用标准GREEDY算法求解，置 $w=s$；然后对{a[2], …, a[n]; v}用标准GREEDY算法求解，如果 $s>w$，那么置 $w=s$；接着对{a[3], …, a[n]; v}用标准GREEDY算法求解，如果 $s>w$，那么置 $w=s$. 依次类推.

过程说明：所求问题的解为 w.

```
procedure GREEDY__A; { Optimization form }
var
    i, j, k:integer; s, u, w:longint;
begin
    k:=0; w:=0;
    for i:= 1 to n do
    begin
        k:=k+1; s:=0; u:=v;
        for j:=k to n do if a[j] ≤ u then
        begin
            s:=s+a[j]; u:=u-a[i]
        end;
        if s > w then w:= s
    end
end;
```

2.3 改进的 Greedy 算法 B

下面 s 的定义和标准的 GREEDY 算法中的 s 相同. 该算法的时间复杂性为 O(n * n).

判定形式: 这个算法的计算过程是这样的, 首先令 $x[1]=0$, 对$\{a[2], \ldots, a[n]; v\}$用标准的 GREEDY 算法求解, 如果 $s=v$, 那么停机; 否则令 $x[2]=0$, 对$\{a[1], a[3], \ldots, a[n]; v\}$用标准 GREEDY 算法求解, 如果 $s=v$, 那么停机; 否则令 $x[3]=0$, 对$\{a[1], a[2], a[4], \ldots, a[n]; v\}$用标准 GREEDY 算法求解, 如果 $s=v$, 那么停机; 依次类推.

过程说明: 如果 $t=$true 那么问题有解, 如果 $t=$false 那么问题无解.

```
procedure GREEDY_B; { Decision form }
var
   i, k:integer; s, u:longint; t:boolean;
begin
   k:=0;
   repeat
      k:=k+1; s:= 0; u:= v;
      for i:= 1 to n do if (i<>k) and (a[i]<=u) then
      begin
         s:=s+a[i]; u:=u-a[i]
      end
   until (s = v) or (k = n);
   t:= (s = v)
end;
```

最优化形式: 首先令 $x[1]=0$, 对$\{a[2], \ldots, a[n]; v\}$用标准 GREEDY 算法求解, 置 $w=s$; 然后令 $x[2]=0$, 对$\{a[1], a[3], \ldots, a[n]; v\}$用标准 GREEDY 算法求解, 如果 $s>w$, 那么置 $w=s$; 接着令 $x[3]=0$, 对$\{a[1], a[2], \ldots, a[n]; v\}$用标准的 GREEDY 算法求解, 依次类推.

过程说明: 所求问题的解是 w.

```
procedure GREEDY_B; {Optimization form }
var
   i, j, k:integer; s, u, w:longint;
begin
   k:= 0; w:= 0;
   for i:= 1 to n do
   begin
      k:=k+1; s:= 0; u:=v;
      for j:= 1 to n do if (j<>k) and (a[j]<=u) then
      begin
         s:=s+a[j]; u:=u-a[j]
      end;
```

```
      if s > w then w:=s
    end
end;
```

2.4 改进的 Greedy 算法 C

这个算法是 Greedy__A 和 Greedy__B 的综合. 组合最优化问题中的实用算法一般都是近似算法，用几个不同的算法处理同一个问题，然后选取效果最佳的一个，是经常用到的技术. 我们能够这样做，一是 Greedy__A 算法和 Greedy__B 算法是基于不同的策略，从结构上看差异较大；二是运行时间大致相同，有相同的时间复杂性.

判定形式：先用 Greedy__A 算法求解，如果 $s=v$，那么停机；否则继续用 Greedy__B 算法求解. 当然二者的顺序也可以颠倒过来.

过程说明：如果 $t=$true，那么问题有解，如果 $t=$false，那么问题无解.

```
procedure GREEDY__C; { Decision form }
var
  i, k:integer; s, u:longint; t:boolean;
begin
  k:= 0;
  repeat
    k:=k+1;  s:=0;  u:=v;
    for i:= 1 to n do if a[i] < u then
    begin
      s:=s+a[i];  u:=u-a[i]
    end
  until (s = v) or (k = n);
  if s <> vthen
  begin
    k:= 0;
    repeat
      k:=k + 1;  s:= 0;  u:=v;
      for i:= 1 to n do if (i<> k) and (a[i] <u) then
      begin
        s:=s+a[i];  u:=u-a[i]
      end
    until (s = v) or (k = n);
  end;
  t:=(s=v)
end;
```

最优化形式：先用 Greedy__A 算法求解，置 $w=s$；再用 Greedy__B 求解，如果 $s>w$，那么 $w=s$. 同样上述两个算法的顺序可以颠倒过来.

过程说明：所求问题的解是 w.

```
procedure GREEDY_C;  { Optimization form}
var
   i,  j,  k:integer;  s,  u,  w: longint;
begin
   k:= 0;   w:= 0;
   for i:= 1 to n do
   begin
      k:=k+1;   s:=0;   u:=v;
      for j:= k to n do if a[j] < u then
      begin          s:=s+a[j];   u:=u-a[j]
      end;
      if s > w then w:=s
   end;
   k:= 0;
   for i:= 1 to n do
   begin
      k:=k+1;   s:=0;   u:=v;
      for j:= 1 to n do if (j<>k) and (a[j] < u) then
      begin
         s:=s+a[j];   u:=u-a[j]
      end;
      if s > w then w:=s
   end
end;
```

3 运行结果及分析

以下是在 ACER 1100／33 PC 上对 100 个例子进行计算得到的结果，其中

OPT: 达到最优解的次数;

MRE: 最大相对误差，单位是 E−6;

ARE: 平均相对误差，单位是 E−6;

DAT: 当 $N=200$ 时，判定型算法的平均运行时间，时间单位是 S[CPU].

OAT: 当 $N=200$ 时最优化型算法的平均运行时间，时间单位是 S[CPU].

表 1 对 100 个例子的计算结果

SCALE	N=100			N=200			N=300			DAT	OAT
ALGORITHM	OPT	MRE	ARE	OPT	MRE	ARE	OPT	MRE	ARE		
GREEDY	2	950.	150.	4	250.	34.	3	90.	12.	.11	.11
GREEDY_A	30	31.	5.6	57	5.9	.52	88	1.4	.063	.21	.32
GREEDY_B	26	54.	5.4	59	3.1	.46	88	1.1	.065	.23	.48
GREEDY_C	41	19.	2.8	80	1.7	.17	97	.41	.012	.33	.69

从以上对100个例子的计算结果中可以看出，改进后的三个Greedy算法GREEDY_A，GREEDY_B和GREEDY_C的各项指标大大优于原来标准的GREEDY算法，而且问题的规模越大，效果就越显著，当然运行的时间也会有所增加．上述结果还启发我们，v的上限固定时，n越大(1)就越可能有解；当n大到一定程度时，再加上其它一些简单的条件，(1)就必定有解，这些将在以后的文章中讨论．

参考文献

1 Papadimitriou C, Steiglitz K. Combinatorial Optimization Algorithms and Complexity. Prentice-Hall, Inc, Englewood Cliffs, N J, 1982

2 Horowitz E, Sahni S. Fundamentals of Computer Algorithms. Pitman Inc Potomac, Md, 1978

3 Garey M, Johnson D. Computers and Intractability: A Guide to the Theory of NP-completeness. San Francisco, Freeman Company, 1979

4 Syslo M, Deo N, Kowalik J. Discrete Optimization Algorithms with Pascal Programming. Prentice-Hall, Inc, Englewood Cliffs, N J, 1983

(上接第79页)

4 结论

CIMS系统的功能集成是一个非常复杂的问题，它不仅关系到企业某个局部环境的调度问题，而且也涉及到整个企业的生产经营活动的综合调度与协调．本文提出的基于黑板处理的CIMS运行时的动态调度系统结构，揭示了CIMS系统运行规律和各功能活动之间的相互关系，可以较好地解决企业里各种生产经营活动的调度问题．

作者准备在实验室里利用国家863计划课题经费的支持，初步建立起一个CIMS系统集成仿真环境，然后在此环境下利用作者提出的结构研究CIMS中的功能集成问题，为解决功能集成打下基础．

参考文献

1 王康华等译．计算机集成制造系统结构CIMS-OSA．华东工学院，1989．12

2 沈阳鼓风机厂CIMS联合设计组．沈阳鼓风机厂CIMS系统初步设计报告．1991．9

3 王富东等．基于黑板模型的智能控制结构．信息与控制；1990；19(2)