找钱问题的动态规划解法*)

贾 驰 王相海
(辽宁师范大学计算机与信息技术学院 大连116029)
(北京大学视觉与听觉信息处理国家重点实验室 北京100871)

摘 要 动态规划算法对许多实际问题是灵活和有效的。本文首先对一类找钱问题进行了分析和讨论,然后给出了该问题的一种动态规划解法,最后对所给算法的复杂性进行了分析。实验结果验证了所提出算法的有效性。

关键词 动态规划,算法,找钱问题,复杂性

The Dynamic Programming Algorithm of the Change Problem

JIA Chi WANG Xiang-Hai
(College of Computer and Information Technology, Liaoning Normal University, Dalian 116029)
(Center of Information Science, National Lab. on Machine Perception of Beijing University, Beijing 100871)

Abstract The dynamic programming algorithm is a flexible and high-efficient method to many problems. In this paper, a kind of change problem is brought up firstly. And then a novel algorithm for this problem based on dynamic programming is proposed. Finally, the complexity of the proposed algorithm is analyzed. Simulation results show it is effective.

Keywords Dynamic programming, Algorithm, Change problem, Complexity

1 引言

动态规划算法作为一种重要的算法设计策略为具有最优子结构性质的实际问题提供了一种重要的解决问题的途径[1]。该策略最早由 Bellman 等人提出[2],它利用最优性原理和所获得的递推关系去求解最优决策序列,从而使问题的计算量急剧下降。利用动态规划策略求解的问题通常可能会有许多可行解,每一个解都对应于一个值,动态规划求解过程希望获得具有最优值的那个解[3]。几年来,人们在这一领域进行了积极的研究,给出了很多具有重叠子问题特性问题的动态规划解法,具体情况可参见文[4,5]。

在现实生活的商品交易中经常涉及到找钱问题,如何设计一个高效的找钱算法对于自动控制,特别是经济领域机器人的发展具有重要意义。本文首先对找钱问题进行了说明,并对该问题采用贪心策略所存在的问题进行了分析,进而对该问题的最优子结构性质进行了证明,在此基础上给出了该问题的递归结构,同时实现了其动态规划算法。最后对所提出算法的复杂度进行了分析和讨论。

2 问题的提出

设有 n 种不同面值的硬币,各硬币的面值存于数组 $T[1:n]$。现要用这些面值的硬币来找钱,可以使用的各种面值的硬币个数不限。问题:对于任意需要找的钱数 j,确定所使用的最少硬币个数以及具体的找钱方案。

3 找钱问题的贪心解法分析

对于找钱问题人们很容易想到贪心算法,比如假设有六种面额分别为50元、20元、10元、5元、2元和1元的货币,现在要找给某顾客86元钱,这时我们会不加思索地拿出一个50元,个20元,一个10元,一个5元和一个1元的货币交给顾客,此时采用这种方法所拿出的货币个数是最少的,这种方法即为贪心算法。贪心算法总是作出在当前看来是最好的选择,其特点简单,因而人们比较喜欢采用。但由于该策略不是从整体最优考虑问题,因而它所做出的选择只是在某种意义上的局部最优选择。比如在上面所举的例子中,如果所给的找钱面值改为1元、5元和11元三种,而要找给顾客的是15元钱,则按照上述贪心算法,找给顾客的面额为:一个11元和四个1元钱,然而找三个5元显然是最好的找法。贪心算法对诸如找钱等问题有时不能得到整体最优解。

4 找钱问题的动态规划解法

4.1 符号约定

(1)设存于数组 $T[1:n]$中的 n 个不同的面值呈递增顺序,找出的钱数 j 满足$1\leqslant j\leqslant L$;

(2)$C(n,j)$表示利用 n 种不同面值的硬币找出钱数为 j 时所用的最少硬币个数,若找不出钱数 j,则约定 $C(n,j)=\infty$;

(3)$P(i,j)(1\leqslant i\leqslant n)$表示按照上述最优值(即 $C(n,j)$)进行找钱所需要的各硬币的个数,即 $P(i,j)$表示当找钱数为 j 时,用到货币面值为 i 的硬币的个数。若 $P(i,j)=0$,则表示没用到面值为 i 的硬币。

4.2 问题的最优子结构性质

结论:对于任意需要找的钱数 j,一个利用 $T[1:n]$中的 n 个不同的面值货币进行找钱的最佳方案为:$P(T(1),j),P(T(2),j),\cdots,P(T(n),j)$,即此时的最少硬币个数 $C(n,j)$为 $\sum_{k=1}^{n}P(T(k),j)$,则 $P(T(2),j),\cdots,P(T(n),j)$一定是利用

*)本文受到国家自然基金项目(60372071)、辽宁省自然基金项目(20032125)、大连市科技基金计划项目和辽宁省高等学校优秀人才支持计划资助。

$T[1:n]$中的n个不同的面值硬币对钱数$j'=j-P(T(1),j)\times T(1)$进行找钱的最佳方案。

证明：假设$P(T(2),j),\cdots,P(T(n),j)$不是最佳方案，不妨假设$P'(T(1),j'),\cdots,P'(T(n),j')$利用$T[1:n]$中的$n$个不同面值的硬币对钱数$j'$进行找钱的最佳方案，其中$P'(T(k),j')(k=1,2,\cdots,n)$表示新方案中对$j'$找钱时所用到的$T(k)$的个数，从而有：

$$\sum_{k=2}^{n}P(T(k),j)>\sum_{k=1}^{n}P'(T(k),j')$$

进一步有：$\sum_{k=1}^{n}P(T(k),j)>P(T(1),j)+\sum_{k=1}^{n}P'(T(k),j')$

从而有：

$$P(T(1),j)+\sum_{k=1}^{n}P'(T(k),j')=P(T(1),j)+P'(T(1),j')+\sum_{k=2}^{n}P'(T(k),j')$$

且：$j=T(1)\times(P(T(1),j)+P'(T(1),j'))+\sum_{k=2}^{n}P'(T(k),j')\times T(k)$

故：$P(T(1),j)+P'(T(1),j'),P'(T(2),J'),\cdots,P'(T(n),j')$也是利用$T[1:n]$中的$n$个不同的面值货币对$j$进行找钱的一个方案，并且其较方案$P(T(1),j),P(T(2),j),\cdots,P(T(n),j)$更加优秀，这与$P(T(1),j),P(T(2),j),\cdots,P(T(n),j)$为最佳方案矛盾。从而上述找钱问题具有最优子结构性质。

4.3 问题的递归结构

以下分几种情况讨论：

(1)当$n=1$时，即只能用一种硬币找钱，硬币的面值为$T[1]$，此时有：

$$C(1,j)=\begin{cases}j/T[1], & j\%T[1]==0;\\ \infty, & j\%T[1]\neq 0;\end{cases}$$

$$P(i,j)=\begin{cases}j/T[1], & i==1\&\&j\%T[1]==0;\\ 0, & \text{其余};\end{cases}$$

(2)当$n>1$时，

① 若$j=T[n]$，即用n种硬币找钱，且第n种硬币面值与找的钱数j相等，此时有：

$C(n,j)=C(n,T[n])=1;P(i,j)=P(i,T[n])=\begin{cases}1,i==T[n];\\ 0,i\neq T[n]\end{cases}$

② 若$j<T[n]$，即第n种硬币面值比所找钱数大，故找钱所使用的硬币只需考虑$T[1:n-1]$即可，故此时有：

$C(n,j)=C(n-1,j);P(T[n],j)=0$。

③ 若$j>T[n]$，即第n种硬币面值比所找的钱数少，故n种硬币都可以用来找钱，此时有：

$$C(n,j)=\min_{1\leqslant k\leqslant n}C(n,j-T[k])+1$$

若当k为$k_0(1\leqslant k_0\leqslant n)$时，$C(n,j)$达到最小值，则有：

$$P(T[k_0],j)=P(T[K_0],j-T[k_0])+1$$

4.4 算法的实现

设数组$T[1:n]$中存放的是n个呈递增顺序的不同面值，所要找的钱数j满足$1\leqslant j\leqslant L$，其中L是由用户输入的；数组$C[j]$表示利用$T[1:n]$找钱数为j时所用的最少硬币个数，即最优值；$P[i][j](1\leqslant i\leqslant n,1\leqslant j\leqslant L)$表示按照上述最优值找钱$j$时用到货币面值为$i$的硬币的个数，即最优解。按照上述递归公式所给出的递推算法如下：

```
#define mmax 10000 // mmax 表示无穷大
void coin(int T[ ],int L,int n)
// 利用 n 个面值为 T[1:n]的硬币，对钱数1～L 进行
// 找钱，输出其最佳找钱方案和所需要的硬币个数
{
int C[100],P[100][100]; // 假设 n 和 L 均不超过100
int i,j,k,aa,bb,minb,flag;
        // minb 用来统计所用硬币个数
for(j=1;j<=L;j++)
{
  if(n>1)
  {
    k=n; flag=0;
    while( k>1 && flag==0 )
    { flag=0;
      if(j==T[k])
      { C[j]=1;
        cout<<"\nj="<<j<<"\t"<<C[j];
        flag=1;
        for(i=1;i<=n;i++)
        { P[i][j]=0;
          P[k][j]=1;
          cout<<"\t T["<<i<<"]="<<P[i][j];
          }
      }
      else if(j<T[k]) k--;
          else
           { minb=C[j-T[k]];
           aa=j-T[k];
           bb=k;
           for(i=1;i<k;i++)
            if(minb>C[j-T[k-i]])
             { minb=C[j-T[k-i]];
               aa=j-T[k-i];
               bb=k-i;
             }
           C[j]=minb+1;
           cout<<"\nj="<<j<<"\t"<<C[j];
           flag=1;
           for(i=1;i<=n;i++)
           { if(C[j]>=10000) P[i][j]=0;
                else
                 { P[i][j]=P[i][aa];
                   P[bb][j]=P[bb][aa]+1;
                   }
             cout<<"\tT["<<i<<"]="<<P[i][j];
           }
        }
      }
      if(j%T[1]!=0&&flag==0)
        { C[j]=mmax; // 找不开
          cout<<"\nj="<<j<<"\t"<<C[j];
          for(i=1;i<=n;i++)
          { P[i][j]=0;
            cout<<"\t T["<<i<<"]="<<P[i][j];
          }
        }
      else if(flag==0)
           { C[j]= j / T[1];
             cout<<"\nj="<<j<<"\t"<<C[j];
             P[1][j]=j / T[1];
             cout<<"\t T[1]= "<<P[1][j];
             for(i=2;i<=n;i++)
             { P[i][j]=0;
             cout<<"\t T["<<i<<"]="<<P[i][j]);
             }
           }
    }
  else
  { if(j%T[1]!=0) C[j]=mmax;
        else C[j]=j/T[1];
    cout<<"\nj="<<j<<"\t"<<C[j];
    P[1][j]=j/T[1];
    Cout<<"\t T[1]= "<<P[1][j];
  }
}
}
```

4.5 算法的实验结果

我们利用如下主调函数对所设计的算法进行了实验。

```
main( )
{ int L,n,T[100];
int i,j;
cout<<"\nPlease input n,L:";
cin>>n>>L;
cout<<"Please input T[1: "<<n<<"]:");
for(i=1;i<=n;i++)
cin>>T[i];
coin(t,L,n);
cout<<endl;
```

}

运行结果如下：

```
Please input n,L:520
Please input T[1:5]:3 6 8 9 13
j=1  10000 T[1]=0 T[2]=0 T[3]=0 T[4]=0 T[5]=0
j=2   1000 T[1]=0 T[2]=0 T[3]=0 T[4]=0 T[5]=0
j=3      1 T[1]=1 T[2]=0 T[3]=0 T[4]=0 T[5]=0
j=4  10000 T[1]=0 T[2]=0 T[3]=0 T[4]=0 T[5]=0
j=5  10000 T[1]=0 T[2]=0 T[3]=0 T[4]=0 T[5]=0
j=6      1 T[1]=0 T[2]=1 T[3]=0 T[4]=0 T[5]=0
j=7  10001 T[1]=0 T[2]=0 T[3]=0 T[4]=0 T[5]=0
j=8      1 T[1]=0 T[2]=0 T[3]=1 T[4]=0 T[5]=0
j=9      1 T[1]=0 T[2]=0 T[3]=0 T[4]=1 T[5]=0
j=10 10001 T[1]=0 T[2]=0 T[3]=0 T[4]=0 T[5]=0
j=11     2 T[1]=1 T[2]=0 T[3]=1 T[4]=0 T[5]=0
j=12     2 T[1]=1 T[2]=0 T[3]=0 T[4]=1 T[5]=0
j=13     1 T[1]=0 T[2]=0 T[3]=0 T[4]=0 T[5]=1
j=14     2 T[1]=0 T[2]=1 T[3]=1 T[4]=0 T[5]=0
j=15     2 T[1]=0 T[2]=1 T[3]=0 T[4]=1 T[5]=0
j=16     2 T[1]=1 T[2]=0 T[3]=0 T[4]=0 T[5]=1
j=17     2 T[1]=0 T[2]=0 T[3]=1 T[4]=1 T[5]=0
j=18     2 T[1]=0 T[2]=0 T[3]=0 T[4]=2 T[5]=0
j=19     2 T[1]=0 T[2]=1 T[3]=0 T[4]=0 T[5]=1
j=20     3 T[1]=1 T[2]=0 T[3]=1 T[4]=1 T[5]=0
```

4.6 算法复杂性分析

(1) 时间复杂性：

由前面算法可知，在计算最优值 $C(j)$ 时，有两次循环 for ($j=1;j<=L;j++$)和 while($k>1$&&flag==0)，若每次都执行循环体里面的 for($i=1;i<k;i++$)，即算法的最坏时间复杂度为 $O(L\times n^2)$；若不执行循环体里面的 for($i=1;i<k;i++$)，即算法的最好时间复杂度为 $O(L\times n)$。

在计算最优解 $P(i,j)$ 时，由于打印出来的是一二维数组，所以多了一层循环，因而最好情况下的时间复杂度为 $O(L\times n^2)$，而最坏情况下的时间复杂度为 $O(L\times n^3)$。

(2)空间复杂性：

过程 coin 中，计算 $C(j)$ 时使用了一个长度为 L 的数组和若干个简单变量作为临时存储单元，故空间复杂性为 $O(L)+O(1)$，即为 $O(L)$。同理计算 $P(i,j)$ 时，用了一个二维数组 $P[L][n]$ 和如果简单变量作为临时存储单元，故此时的空间复杂度是 $O(L\times n)$。

结束语 本文通过理论上的分析，说明了贪心算法在解决找钱问题上的不足，并进一步证明了该问题可用动态规划求解的最优子结构性质，在此基础上给出了问题最优值和最优解的递归关系以及问题求解的具体算法，最后对算法的时间和空间复杂性进行了分析和讨论，算法的复杂度是比较令人满意的。实验结果验证了方法的有效性。

参 考 文 献

1 Katoen J P. Dynamic Programming Algorithm. http://fmt. cs. utwente. nl/courses/adc/lec8. pdf 2002. 10
2 Bellman R E. Dynamic Programming. Princeton University Press, 1957
3 余祥宣，崔国华，邹海明. 计算机算法基础. 武汉：华中科技大学出版社，2000
4 Dettinger M. Developing a Dynamic Programming algorithm. http://www. aduni. org/courses/algorithms/handouts/Reciation_06. html 2001. 2
5 Dawes R. Thoughts on Dynamic Programming. http://www. cs. queensu. ca/home/cisc365/1998Dawes/dynprognotes. html 1998
6 王晓东. 计算机算法设计与分析. 北京：电子工业出版社，2001

(上接第186页)

由于这里能够系统地进行设计，就必须重点考虑以下几个问题：

1)在资源库中建立跨学科的知识描述机制。基于传统的资源库，常见的协作学习只是对应于某一领域知识的协作学习，系统的知识框架也是针对某一特定学科的。这就大量失去参加学习的角色(教师、学习者、泛化的角色)的个性，使协作流于形式化、作品化。其主要原因之一，是由于系统在设计时没有一个跨学科的知识描述机制。

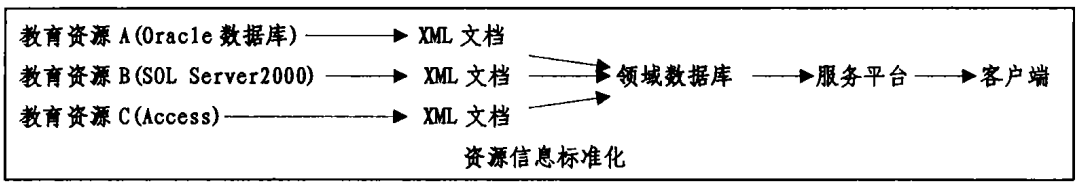

图2

2)建立协同对象互操作语义视图。为了使协作学习具有个性化、避免形式化和作品化，还应在协同学习环境里，建立协同对象互操作语义视图。这是因为协作学习的角色可能分布在不同地方甚至是不同国家，他们所用的物理环境也可能是异构的。因此，这些对象之间就存在着语义差别，使得对象之间的互操作变得困难。目前的研究大都是对多信息源采用统一的对象描述及视图来处理协作对象的互操作问题。在此基础上，建立互操作语义视图，可能是效率更高的思路。

3)建立可以基于协同学习目标而进行资源重组的资源重组机制。这是有效利用现有资源、使协同学习真正实用的主要途径，也是建立跨学科的知识描述机制和建立协同对象互操作语义视图的目的。

结束语 以上讨论了基于资源的计算机支持的协作式学习 CSCLBR 的描述、结构和实现思路，并简要介绍了我们的部分工作。CSCLBR 是一种新的探索，特别是资源重组和寻求合适的 CSCL 模式更是今后必须面对的挑战。有关文献已经从“网格”、“主体”、“本体”、“对等网”……和学习理论等方面开展研究，这将有力推动 CSCLBR 的研究和实际应用。

参 考 文 献

1 李吉桂，李小文，邹家炜. 计算机支持的协同学习(CSCL)系统的结构、工作模式和开发策略. 计算机科学，1998，25(4)
2 网格环境中资源发现机制的研究. http://grid. cs. tsinghua. edu. cn/grid/paperppt/YF200312/Q1481. pdf
3 网络异步-学习管理系统. http://www. online-edu. org/ellkm/articles/ta/53. html
4 王桂玲. CSCL 系统和工具的分类与比较. http://www. online-edu. org/member/article/648. html
5 黄荣怀. CSCL 的理论与方法. 北京师范大学