兑换零钱问题的动态规划算法研究*

严华云[1,2]

(1. 湖州师范学院 信息工程学院, 浙江 湖州 313000; 2. 同济大学 电子与信息工程学院, 上海 201804)

摘 要: 兑换零钱问题是一个求解组合优化的问题。首先对兑换零钱问题进行了分析, 证明了该问题满足动态规划的最优化原理, 并给出了其动态规划解法; 然后对本算法进行了时间复杂性和空间复杂性分析, 得到时间复杂性由通常的动态规划算法的 $O(Mn^2)$ 提高到本算法的 $O(n^2)$, 空间复杂性由通常的动态规划算法的 $O(Mn)$ 提高到本算法的 $O(n^2)$, 因此效率有了较大提高。最后通过实验对算法进行验证, 证明了算法的高效性。该算法可以广泛应用于自动售货机。

关键词: 动态规划; 兑换零钱问题; 算法复杂性

中图分类号: TP311.11 文献标志码: A 文章编号: 1001-3695(2007)12-0088-03

Research on money change problem through dynamic programming

YAN Hua-yun[1,2]

(1. School of Information & Engineering, Huzhou Teachers College, Huzhou Zhejiang 313000, China; 2. College of Electronics & Information Engineering, Tongji University, Shanghai 201804, China)

Abstract: Money change problem is a combinatorial optimization problems. Firstly analyzed money change problem, and proposed a dynamic programming algorithm. Then, analyzed the space complexity and time complexity of the algorithm, enhanced the space complexity from $O(Mn^2)$ of general algorithm to $O(n^2)$ of this algorithm, enhanced the time complexity from $O(Mn)$ of general algorithm to $O(n^2)$ of this algorithm, thus improved the efficiency much. Finally tested the algorithm through simulation, and found the algorithm was high-efficient. This algorithm can be used in the field of automatic sales.

Key words: dynamic programming; money change problem; complexity algorithm

动态规划是求解决策过程的有效最优数学方法之一[1-3]。它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[4,5]。该策略最早由 Bellman提出[4,6], 它利用最优性原理和所获得的递推关系去解最优决策序列, 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, 每个解均对应一个值, 动态规划求解过程希望获得具有最优值的那个解[7]。多年来, 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定问题的动态规划解法。详细情况可参见文献[8,9]。

兑换零钱问题有其广泛的应用前景, 如应用到自动售货机上进行零钱兑换。被称为“永不下班的超级营业员”的自动售货机在中国特别是发达国家越来越普及, 但目前自动售货机出售商品时, 机器只能接收硬币和小额纸币, 影响到消费者的选择[10]。如何解决该问题, 其中关键的技术包括对大额纸币的零钱兑换问题。由于运行在自动售货机上, 设计一个高效的兑换零钱算法对于本问题相当重要。本文首先分析了用贪心算法解决该问题。由于贪心算法固有的缺点决定了其并不总能解决最优问题, 还需要进行证明[7]其算法的最优性。比如说世界上通行的货币体制为十进制, 这种货币体制的零钱兑换问题用贪心算法兑换可以证明是最优的。但有少数的币制不是十进制的就未必能用贪心算法兑换, 如对于以前的英国货币单位是 1英镑等于 20先令, 1先令等于 12便士; 同样, 以前法国的币制也存在这样的问题。因此本文对该问题用动态规划算法加以解决, 以避免非十进制换算的币制体系不能用贪心算法解决。其间证明了兑换零钱问题可以进行动态规划, 给出了该问题的最优子结构性质的证明以及该问题的递归结构(重叠子问题); 同时用 C++实现了其动态规划算法。最后对该算法的复杂度进行了分析。

兑换零钱问题的贪心解法

1. 兑换零钱问题的问题表述

一个货币系统有 n种不同面值的钱币, 各种钱币的面值分别为 $T_1, T_2, \cdots, T_n$。其中 $T_1=1$。要怎样兑换价值为 M的钱(即面值为 M的兑换零钱问题), 使钱币的数目最少。形式地说, 要让量$\sum_{i=1}^{n} x_i$在约束条件$\sum_{i=1}^{n} x_i T_i = M$下极小。其中: $x_1, x_2, \cdots, x_n$是非负整数, 表示该种币值(T_i)的个数; 如果其中 $x_i=0$, 则指没有该种币值(T_i)。

2. 兑换零钱问题的贪心解法分析

对于兑换零钱问题的贪心算法可以这样实现:

假设 $T_1<T_2<\cdots<T_n$,

输入: 需要兑换零钱的数 M, $T_1, T_2, \cdots, T_n$;

输出: 根据 $X[j]$的值输出 T_i的个数。

收稿日期: 2006-11-12; 修返日期: 2007-10-18 基金项目: 国家自然科学基金资助项目(60573056); 浙江省自然科学基金资助项目(Z106335, Y105090); 湖州市科技计划资助项目(2006GG03, 2006YG01, 2007YZ08)

作者简介: 严华云(1972-), 男, 讲师, 博士研究生, 主要研究方向为信息检索、数据挖掘、信息安全、计算机辅助设计(yanhy@hutc.zj.cn).

```
while(M>0) 从 Tn→T1
    if Ti<=M
    { Xi←M/Ti (M/T表示相除取整, 1≤i≤n)
      M←M-Ti*Xi }
end while
```

从该贪心算法来看,其效率很高。但贪心算法的关键不是写出算法,而是要证明该算法的确是最优解。事实上,如果钱币的面值分别为 1、5、7和 11分,要兑换 15分的钱,如用贪心算法的解会是 1个 11分,4个 1分;但最优解应为 3个 5分。因此对于不同的钱币系统,使用贪心算法有时并不能得出整体最优解。

兑换零钱问题的动态规划算法

. 符号说明

a)设存于数组 T[ ]中的 n个不同的面值呈递增顺序,找零的钱数 L。

b) C(n, j)表示利用 n种不同面值的硬币找出钱数为 j(1≤j≤L)时所用的最少硬币个数,若找不出钱数 j,则约定 C(n, j)=∞。

c) P(i, j)(1≤i≤n)表示按照上述最优值(即 C(n, j))进行兑换零钱所需要的各硬币个数,即 P(i, j)表示当兑换零钱数为 j时,用到货币面值为 i的钱币个数。若 P(i, j)=0,则表示没有用到面值为 i的硬币。

. 兑换零钱问题的最优子结构性质

兑换零钱问题的最优子结构表述:对于任意需要找的钱数 j,一个利用 T[n]中的 n个不同面值钱币进行兑换零钱的最佳方案为 P(T(1), j), P(T(2), j), …, P(T(n), j),即此时的最少钱币个数 $C(n, j)=\sum_{k=1}^{n} P(T(k), j)$,则 P(T(2), j), …, P(T(n), j)一定是利用 T[n]中 n个不同的面值钱币对钱数 j=j-P(T(1), j)×T(1)进行兑换零钱的最佳方案。

证明　假设 P(T(2), j), …, P(T(n), j)不是最佳方案,不妨设 P′(T(1), j), P′(T(2), j), …, P′(T(n), j)是对于 T[n]中 n个不同面值的钱币对钱数 j进行兑换零钱的最佳方案。其中 P′(T(k), j)(k=1, 2, …, n)表示新的方案中对 j兑换零钱时所用到的 T(K)的个数,因此有

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

从而有 $\sum_{k=1}^{n} P(T(k), j) > P(T(0), j)+\sum_{k=1}^{n} P'(T(k), j)$;另有 $P(T(1), j)+\sum_{k=1}^{n} P'(T(k), j) = P(T(1), j)+P'(T(1), j)+\sum_{k=2}^{n} P'(T(k), j)$;并且有 j=T(1)×[P(T(1), j)+P′(T(1), j)+$\sum_{k=2}^{n}$P′(T(k), j)]×T(k)。P(T(1), j)+P′(T(1), j), P′(T(2), j), …, P′(T(n-1), j)也是利用 T[n]中 n个面值货币对 j进行兑换零钱的一个方案,并且比方案 P(T(1), j), P(T(2), j), …, P(T(n), j)更加优化。这与 P(T(1), j), P(T(2), j), …, P(T(n), j)为最佳方案矛盾。因此,最优子结构性质得证。

. 兑换零钱问题的递归结构(重叠子问题)

a)当 n=1时,即只能用一种钱币兑换零钱,钱币的面值为 T[0],有

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

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

b)当 n>1时,

若 j<T[n],即第 n种钱币面值比所兑换零钱数小,因此有 $C(n, j)=\min_{1\le k\le n}\{C(n, j-T[k])+1\}$。当 k为 k_0(1≤k_0≤n)时,C(n, j)达到最小值,有 P(T(k_0), j)=P(T(k_0), j-T(k_0))+1。

若 j=T[n],即用 n种钱币兑换零钱,第 n种钱币面值与兑换零钱数 j相等,此时有 C(n, j)=C(n, T[n])=1;P(i, j)=P(i, T[n])=$\begin{cases} 1, & i=T[n] \\ 0, & i\neq T[n] \end{cases}$。

若 j<T[n],即第 n种钱币面值比所兑换零钱数大,因此兑换零钱只需考虑前 n-1种钱币即可,故有 C(n, j)=C(n-1, j),且 P(T(n-1), j)=0。

从以上讨论可知该问题具有重叠子问题性质。

. 算法实现

设数组 T[ ]中存放的是 n种钱币递增的不同面值,所要找的钱数为 M,M由用户输入;数组 C[j]表示利用数组 T[n]兑换零钱数为 j时所用的最少钱币个数,即最优值;P[j][j](1≤j≤n)表示按照上述最优值兑换零钱 j时用到钱币面值为第 i种钱币的个数。根据递归算法有如下动态规划解法:

```
#define max 10000 //用 max表示无穷大
void money_change(int T[ ], int M, int n)
{ int C[100], P[100][100];
int i, j, k, a, b, min, flag;
for(j=1; j<=M; j++)
  { if(n>1)
    { k=n; flag=1;
    while(k>1&&flag==1)
    { flag=1;
    if(j==T[k])
    { C[j]=1;
    if(j==M) cout<<"\nj="<<j<<"\t"<<C[j];
    flag=0;
    for(i=1; i<=n; i++)
      { P[j][i]=0
      P[k][j]=1;
      if(j==M) cout<<"\tP["<<i<<"]="<<P[j][i];
      }
    }
    else if(j<T[k]) k--;
      else{ min=C[j-T[k]];
      a=j-T[k];
      b=k;
      for(i=1; i<k; i++)
      if(min>C[j-T[k-i]])
      { min=C[j-T[k-i]];
      a=j-T[k-i];
      b=k-i;
      }
      C[j]=min+1;
      if(j==M) cout<<"\nj="<<j<<"\t"<<C[j];
        flag=0
        for(i=1; i<=n; i++)
      { if(C[j]>=1000) P[j][i]=0;
        else P[j][i]=P[j][a];
```

```
                P[ k][ j] = P[ k][ a] + 1;
              }
            if( j == M) cout<< " \t[ "<< i<< "] = "<< P[ i][ j];
                }
          }
      }
    if( j% T[ 1] != 0&& flag== 1)
    { C[ j] = max;
    if( j== M) cout<< " \nj"<< i<< " \t"<< C[ j];
    for( i= 1; i<= n; i++)
      { P[ j][ j] = 0;
        if( j== M) cout<< " \t[ "<< i<< "] = "<< P[ i][ j];
      }
    }
    else if( flag== 1)
    { C[ j] = j/ T[ 1];
    if( j== M) cout<< " \nj"<< i<< " \t"<< C[ j];
    P[ 1][ j] = j/ T[ 1];
    if( j== M) cout<< " \t[ 1] = "<< P[ 1][ j];
    for( i= 2; i<= n; i++)
      { P[ i][ j] = 0;
        if( j== M) cout<< " \t[ "<< i<< "] = "<< P[ i][ j];
      }
      }
    }
    else
    { if( j% T[ 1] != 0) C[ j] = max;
     else C[ j] = j/ T[ 1];
    if( j== M) cout<< " \nj"<< i<< " \t"<< C[ j];
    P[ 1][ j] = j/ T[ 1];
    if( j== M) cout<< " \t[ 1] = "<< P[ 1][ j];
    }
  }
}
```

. 算法的实验验证

```
#include< iostream. h>
void main( )
{ int i, j, M, n, T[ 100];
cout<< " please input n, M: ";
cin>> n>> M;
cout<< "please input from T[ 1] to T[ "<< n<< "] : \n";
for( i= 1; i<= n; i++)
cin>> T[ i];
money_change( T, M, n);
cout<< endl;
}
```

实验结果如下:

实验 1

```
please input n, M: 4 97(有 4种钱币, 对 97进行兑换)
please input from T[ 1] to T[ 4] :
1 5 9 13
j= 97 9 T[ 1] = 0 T[ 2] = 2 T[ 3] = 1 T[ 4] = 6
```

表示钱币有 4种, 分别为 1、5、9、13。当对 97进行兑换零钱时可以找出 6个 13、1个 9和 2个 5, 共 9个钱币。如用贪心算法则应为 6个 13、1个 9和 2个 5, 共为 9个钱币。贪心算法与动态规划都是最优。

实验 2

```
please input n, M: 4 88(有 4种钱币, 对 88进行兑换)
please input from T[ 1] to T[ 4] :
1 5 7 13
j= 88 8 T[ 1] = 0 T[ 2] = 2 T[ 3] = 0 T[ 4] = 6
```

表示钱币有 4种, 分别为 1、5、7、13。当对 88进行兑换零钱时可以找出 6个 13和 2个 5, 共 8个钱币。如用贪心算法则应为 6个 13、1个 7和 3个 1, 共为 10个钱币, 显然不是最优。

实验 3

```
please input n, M: 4 76(有 4种钱币, 对 76进行兑换)
please input from T[ 1] to T[ 4] :
1 5 7 11
j= 76 8 T[ 1] = 0 T[ 2] = 2 T[ 3] = 0 T[ 4] = 6
```

表示钱币有 4种, 分别为 1、5、7、11。当对 76进行兑换零钱时可以找出 6个 11和 2个 5, 共 8个钱币。如用贪心算法则应为 6个 11、1个 7和 3个 1, 共为 10个钱币, 显然不是最优。

从以上实验可以看出, 贪心算法并不能总是得出最优, 因此用动态规范算法具有普适性。

. 复杂性分析

2.6.1 时间复杂性

从上面算法可知, 最优值 C[j]的计算过程中, 最外层为循环 for(i= 1; i<= M; i++)嵌套着 while(k> 1&& flag== 0)循环, 而 while(k> 1&& flag== 0)循环中又嵌套着三个并列的 for 循环。因此本算法最坏情况下的复杂度是 $O(M\times n^2)$; 最好的情况当然是里面 for循环的条件不满足而不执行, 此时的复杂度为 $O(M\times n)$。其中: M表示需要兑换的零钱数, 对于 M来说, 该值一般不是很大, 对于钱币来说, M会小于 100元, 即 10 000分; n表示钱币的种类, n值一般不会很大, 如钱币总的有 13种(从 1分, 2分, …, 100元)。经过以上分析, 如是最坏情况时的复杂度应为 $O(M\times n^2)$, 则该值对于内存和运行速度较小的自动售货机等的应用前景则不会很好。但本算法中的递归结构在 M> T[n]时, 有 $C(n, j) = \min_{1\le k\le n} C(n, j- T[k]) + 1$。可见对于钱币 j= M, 求 C(n, j)时, 并不要求对从 $1\le i\le j$ 的所有情况都要求 $C(n, j) + 1$, 而是只求 $C(n, j- T[k]) + 1$, 其中: $1\le k\le n$。钱币一般只有 13种左右, 因此其效率大为上升。最坏的情况下需要执行 $O(M\times n^2) = O(n^2)$, 而 M小于 100元即 10 000分, 远大于 n。本算法的动态规划算法的时间复杂性比该问题的一般动态规划算法的效率要好得多。该算法的时间复杂性是 10^3数量级的, 对于应用于自动售货机等运行速度较慢的机器来说是不成问题的。

2.6.2 空间复杂性

从上面算法可知, 用到了三个数组, 分别为 T[n], C[j], P[j][j]。其中: $i\le n, j\le M$。空间复杂性主要由 P[j][j]决定, 为 $O(M\times n)$。P(, i, j)中的 i指的是 T[n]中的值, 对于钱币来说一般 n为 13左右。该算法的空间复杂度为 $O(M\times n) = O(n^2)$, 而 M小于 100元即 10 000分, 远大于 n。该算法动态规划的空间复杂性比该问题的一般动态规划的效率要好得多。该算法的空间复杂性为 10^2数量级, 这对于应用到小内存的自动售货机来说是没有任何问题的。

通过上面的时间复杂度和空间复杂度分析, 可以得出兑换零钱问题的动态规划算法的效率是相当高的, 可以应用到即使是小内存、低速度的自动售货机上。因此本算法有广阔的应用前景。

结束语

本文对兑换零钱问题进行研究。首先分析了其贪心算法的不足之处; 然后给出了动态规划解法; 在此过 (下转第 93页)

仅训练全连接结构权值的 BP算法、PSO算法、SW-GA算法以及本算法的测试误差、神经网络连接数与训练阶段的 CPU运行时间比较如表 1所示。

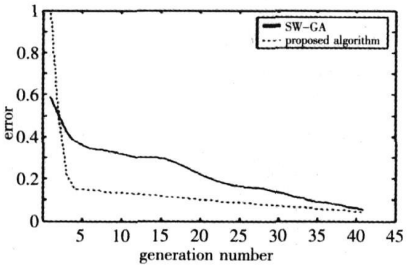

图 1　网络训练算法最优均方误差曲线

表 1　神经网络性能比较

性能指标	BP	PSO	SW-GA	本文算法
测试误差	0.622 7	0.433 2	0.452 6	0.329 4
网络连接数	56	56	38	35
CPU时间 / s	6.125	3.533	5.575	4.123

由表 1可知, 本文算法对结构的优化提高了神经网络的故障模式识别能力, 表明删除的网络连接为冗余连接, 同时也验证了冗余连接对神经网络性能的影响。由实验结果还可以看出, 本文算法与遗传算法相比, 训练效率得到了较大提高, 在训练时间接近时, 网络的诊断能力显著增强; 若达到同样误差目标, 采用粒子群优化算法收敛所需的训练迭代次数明显降低。由此可见, 本文算法不仅使训练的收敛速度大大提高, 且其训练的神经网络性能也得到了增强。需要指出, 尽管对结构的优化增加了本文算法训练阶段的复杂度, 但是经过连接结构优化的神经网络在实际应用中信息处理效率得到提高。经过连接结构优化的神经网络尤其适合大规模数据的实时处理。

结束语

针对神经网络中的冗余连接不仅会降低神经网络的处理速度, 而且大量的冗余连接甚至会影响神经网络性能的情况, 本文提出了综合利用粒子群优化算法和离散粒子群优化算法同时优化前向神经网络结构和参数的新方法。该算法在训练神经网络权值的同时删除其冗余连接, 实现连接结构的优化。在旋转机械故障诊断神经网络上的试验数据表明, 这种方法不但能够有效地优化神经网络的结构, 而且还能提高训练效率。与训练全连接结构权值的 PSO相比, 算法在保证分类正确率的同时减少了连接数目; 与 BP及遗传算法相比, 在提高分类误差精度的同时加快了训练收敛速度。

参考文献:

[1] YANG J M, KAO C Y. A robust evolutionary algorithm for training neural networks[J]. Neural Computing and Application, 2001, 10(3): 214-230.

[2] KENNEDY J, EBERHART R C. Particle swarm optimization[C]// Proc of IEEE International Conference on Neural Networks. Piscataway: IEEE Press, 1995: 1942-1948.

[3] VOSS M S, FENG Xin. Arma model selection using particle swarm optimization and AIC criteria[C]// Proc of the 15th Triennial World Congress. Barcelona, Spain: IFAC, 2002.

[4] PARSOPOULOS K E, VRAHATIS M N. Particle swarm optimization method in multiobjective problems[C]// Proc of ACM Symposium on Applied Computing. New York: ACM Press, 2002.

[5] BERGH F van den, ENGELBRECHT A P. Cooperative learning in neural networks using particle swarm optimizers[J]. South African Computer Journal, 2000, 26: 84-90.

[6] PARSOPOULOS K E, VRAHATIS M N. Recent approaches to global optimization problems through particle swarm optimization[J]. Natural Computing, 2002, 1(2-3): 235-306.

[7] OURIQUE C O, BISCAIA E C, PINTO J J C. The use of particle swarm optimization for dynamical analysis in chemical processes[J]. Computers and Chemical Engineering, 2002, 26: 1783-1793.

[8] KENNEDY J, EBERHART R C. A discrete binary version of the particle swarm algorithm[C]// Proc of Conf on Systems, Man, and Cybernetics. Piscataway: IEEE Press, 1997: 4104-4108.

[9] EBERHART R C, SHI Y. Comparing inertia weights and constriction factors in particle swarm optimization[C]// Proc of Congress on Evolutionary Computing. Piscataway: IEEE Press, 2000: 84-89.

[10] STACEY A, JANCIC M, GRUNDY J. Particle swarm optimization with mutation[J]. Evolutionary Computation, 2003(2): 1425-1430.

(上接第 90页)程中对动态规划算法的充要条件(最优子结构和重叠子问题性质)进行了证明, 给出了算法的递归结构, 并用 TC++环境下的程序进行验证。最后对时间和空间复杂性进行了分析, 算法的复杂性令人满意。实验证明了本算法的有效性。该算法可以广泛应用, 如可以应用到自动售货机上。

参考文献:

[1] 费蓉, 崔杜武. 中国邮递员问题的动态规划算法研究[J]. 计算机研究与发展, 2005, 42(2): 294-299.

[2] 文贡坚, 周秀芝. 基于视差点的大遮挡检测和立体匹配方法[J]. 软件学报, 2005, 16(5): 708-717.

[3] 杨娟, 邱玉辉, 李建国, 等. 一种应用部件可动态规划的 MA模型[J]. 计算机研究与发展, 2005, 42(5): 830-834.

[4] DREYFUS S. Richard Bellman on the birth of dynamic programming[EB/OL]. (2002). http://www.eng.tau.ac.il/~amir/cd/050/1526-5463-2002-50-01-0048.pdf.

[5] 祝远新, 徐光祐, 俞志和. 基于表观的动态孤立手势识别[J]. 软件学报, 2000, 11(1): 54-61.

[6] COOPER R. Dynamic programming an overview[EB/OL]. (2000-02). http://econ.bu.edu/faculty/cooper/dynprog/introlect.pdf.

[7] ALSUWAIYEL M H. Algorithms design techniques and analysis[M]. Beijing: Publishing House of Electronics Industry, 2003: 203-204.

[8] 王晓东. 计算机算法设计与分析[M]. 北京: 电子工业出版社, 2001: 43-44.

[9] NIST. Dynamic programming[EB/OL]. (2005). http://www.nist.gov/dads/HTML/dynamicprog.html.

[10] 叶国际, 吕惠敏. 外商看好我国的自动售货机市场[EB/OL]. (2002-05-24). http://www.cima.org.cn/dzkq/p0703.htm.