# 期末算法分析

多属性限制背包问题

**问题描述：**给定n个物品及其价值，每个物品有m个属性，给定背包对每个属性的限制条件，求出背包能装载的最大价值（所有数据动态输入）。

**问题分析：**我认为，这是一个01背包问题的变种，将限制条件由一个变成了多个。但是其解题的核心没有变，对一个物品来说，还是只有放和不放两种选择，我们要探讨的依旧是如何找出放那个物品最优的问题。我们依然可以把每个属性看成是一个01背包问题，只不过每个属性之间是关联的而已。在这里，我选择用分支限界法来解决它。对于一个传统的01背包问题，我们考虑的核心在于，假设放入第i个物品，那么背包的价值是多少，容量还剩多少，以及把第i个物品取出，放入第i+1个物品，或者之后的物品，那么它的收益是如何变化的。

**程序实现：**首先，我是用C语言来做的，那么对于文件给定的数据格式我们是需要变换一下的，因为C语言的数组只能横向操作，即数组只能对其每一行进行操作，所以我们需要定义一个函数，将每个属性的限制放在每行属性后面。

同时需要划分一个空间来将数据以二维数组的方式存下。

接着就是算法的核心——对数据的处理。对于一个物品放或不放，我们如何判断其是否是最优的呢？其实我们可以先这么思考：将物品一个个放入（当然也可以按性价比从大到小放——单位属性内价值最大）当背包只能放入i或者i+1后的若干个物品的时候，我们对他们的值进行一个考量，若：i+1个物品全放，都不如i的价值大，那自然放i，反之则放i+1。那么，我们不妨做一个估测函数把背包问题的理想最大值（假设物品可拆，在符合限制情况下，将性价比最高的物品从大到小相加——价值与属性比之和最大）算出来，让程序在比较的时候有个参考，比较出放入物品i和i+1的值谁更优。当然，正常形况下的最优值是比理想值小的，但是理想值的作用，主要是作为程序的剪枝限界的标准。

double maxBound(int t){

double bound = c\_value;

int \*left = (int\*)malloc(sizeof(int) \* nUM\_attr);

int i,j;

for(i = 0; i<nUM\_attr; i++)

\*(left + i) = \*(limitation + i) - \*(c\_limit+ i);

while(t < nUM && compare(left, t)){

minus(left, t);

bound += value[t];

t++;

}

if(t < nUM){

bound += (value[t]/attribute[0][t]) \* left[0];

}

return bound;

}

上面这个方法，就是实现一个理想最大值的预估，在放入物品的过程中，理想最大值不断的向实际最优值靠拢，在这一过程中实现剪枝限界。

当然，虽然我们说多属性限制的背包跟传统01背包的差别不大，但还是有差别的。那就是每一个属性之间是有联系的，对一个物品的选择是，所有属性都满足才能选择。那么，我们就引入一个数组的比较，以实现我第一个属性选择的结果也是我第二个属性选择的结果的最优才行。

int compare (int op, int \*l, int \*target){

int i;

if(op == 0){

for(i = 0;i<nUM\_attr;i++){

if(l[i] < target[i])

return 0;

}

}

else if(op == 1){

for(i = 0;i<nUM\_attr;i++){

\*(l+i) -=\*(target+i);

}

}

else if(op == 2){

for(i = 0;i<nUM\_attr;i++){

l[i]+= target[i];

}

}

return 1;

}

在上面的程序中，要比较两个数组，看第二个数组中的每一个元素是否都大于前者，只有都满足条件，才能选择。否则不能。这样，就是实现了属性的关联选择的问题。

mkp:分支限界实现

接着就是将这两个核心之一的组建运用起来：

double mkp(){

int i = 0,j;

double upbound = maxBound(i);

while(1){

int \*c\_weight = (int\*)malloc(sizeof(int)\*nUM\_attr);

memset(c\_weight,0,sizeof(int)\*nUM\_attr);

for(j = 0;j<nUM\_attr;j++){

c\_weight[j] = attribute[j][i] + c\_limit[j];

}

if(do\_something(0, limitation,c\_weight)){

if(c\_value + value[i] > bestv)

bestv = c\_value+ value[i];

addLiveNode(upbound,c\_value + value[i], c\_weight, i + 1);。

}

upbound = maxBound(i + 1);

if(upbound>= bestv)

addLiveNode(upbound,c\_value, c\_limit, i + 1);

else{

free(c\_limit);

}

if(stact\_empty())

return bestv;

headNode \*node = stack-pop();

C\_limit = node->constrain;

c\_value = node->value;

upbound = node->upbound;

i = node->level;

free(node);

node = NULL;

}

}

上面的代码是分支限界的核心部分，其实也是调用了我们前面说的选择物品的规则，以及属性的关联的整合而已。

**时间复杂度的计算：**在这里，因为我们是用二叉树来实现节点的选择，2的n次方远大于n的2次，所以，其时间复杂度最坏可能是有O（）这种情况是，数据过于特殊，限界条件无法用上，程序需要遍历所有结果的情况。而其平均复杂度为：O（）。