第16卷 第4期
2006年4月

计算机技术与发展
COMPUTER TECHNOLOGY AND DEVELOPMENT

Vol. 16 No. 4
Apr. 2006

基于 Matlab 的 0－1 背包问题的动态规划方法求解

王 乐，王世卿，张静乐
（郑州大学 信息工程学院，河南 郑州 450052）

摘 要：背包问题是经典的 NP－hard 组合优化问题之一，在经济管理、资源分配、投资决策、装载设计等领域有着重要的应用价值。文中用动态规划方法解决 0－1 背包问题，通过在 Matlab6.5 环境下对其算法进行测试和与其他方法对比分析，表明应用该方法可节省大量的计算时间，因而具有更高运行效率。

关键词：0－1 背包问题；DP 算法；分治法；递归法；NP 难问题；Matlab
中图分类号：TP301.6 **文献标识码**：A **文章编号**：1005－3751(2006)04－0088－02

DP Algorithm of Solving 0－1's Knapsack Problem Based on Matlab

WANG Le, WANG Shi-qing, ZHANG Jing-le
(College of Information Engineering, Zhengzhou University, Zhengzhou 450052, China)

Abstract: The knapsack problem is a classic NP－hard problem in the combinational optimization. It is valuable in many fields such as resource assignment, investment, decision and loading design. This paper solved the 0－1 knapsack problem by DP algorithm, and test the algorithm in Matlab6.5. The algorithm shows its superiority after comparing with other methods.

Key words: 0－1knapsack problem; DP algorithm; dividing－and－conquering; recursive algorithm; NP－hard problem; Matlab

0 引 言

背包问题是一个典型的 NP－hard 问题，求解背包问题有着广泛的应用前景，在学术上，可以解决 0－1 整数规划问题或某类可归纳为 0－1 背包问题的子问题；在实际应用中，对于解决资源分配、投资决策、预算控制、项目选择和货物装运等问题，其规划方法优劣将直接影响到运作效率与成本[1]。因此，对该问题的研究倍受人们关注。文中在对多种不同的求解方法如贪婪算法、回溯法、分枝－界限法等进行分析研究的基础上，采用动态规划算法解决此类问题。通过在 Matlab6.5 环境下对其算法进行测试，并与其他解决背包问题的方法进行分析对比，表明动态规划方法以空间换时间，对一个问题或多次出现的相同问题仅需解决一次，因此可节省大量的计算时间而具有更高运行效率的优越性。

1 问题描述

背包问题是一个典型的 NP 难问题，它的数学模型实际上是一个 0－1 规划问题。0－1 背包问题是指有不同价值、不同重量的物品 n 件，求从这 n 件物品中选取一部分物品且对每一物品，要么选，要么不选，满足被选物品的总重量不超过背包指定的限制重量且达到被选物品的价值总和最大的问题。如果所有物品的重量之和小于背包的容量，则问题极其简单，所得利益也就是所有物品的价值之和，而实际问题往往是背包的容量小于物品的重量之和[2]。用形式化方法可对该问题描述为：

设 W 为背包的容量，n 个物品的重量组成一向量 $(w_1, w_2, \cdots, w_n)$，其价值组成另一向量 $(v_1, v_2, \cdots, v_n)$，$W>0, w_i>0, v_i>0 (1 \leqslant i \leqslant n)$。要找出另一 n 元向量 $(x_1, x_2, \cdots, x_n)$，$x_i \in \{0,1\}$，$1 \leqslant i \leqslant n$，1 表示选，0 表示不选该物品。

由此，0－1 背包问题要求：$\mathrm{Max}\sum_{i=1}^{n} v_i x_i$，且满足以下两个约束条件：

(1) $\sum_{i=1}^{n} w_i x_i \leqslant W$

(2) $x_i \in \{0,1\}$ $\quad 1 \leqslant i \leqslant n$

也就是说对 0－1 背包问题，可以通过求出变量 $x_1, x_2, \cdots, x_n$ 的一个决策序列来得到它的解，而对变量 x_i 的决策就是决定它是取 0 值还是取 1 值。

2 采用的动态规划方法

对解决复杂问题虽然不能简单地把它分解成几个子问题，但可以分解出一系列的相关子问题。因简单地采用把大问题分解成子问题，并综合子问题的解导出大问题的解的方法，问题求解耗时会按问题规模呈幂级数增加。为节约重复求相同子问题的时间，文中引入一个动态变化的数组，不管每一次对子问题求得的解是否对最终解有用，

收稿日期：2005－07－09

作者简介：王 乐(1981－)，女，河北安新人，硕士研究生，研究方向为决策支持与企业信息化；王世卿，硕士研究生导师，研究方向为电子商务、数据挖掘。

都把所有子问题的解存于该数组中。对于重复出现的子问题,只有在第一次遇到时才给预求解,并把其答案存放在该数组中保存起来,以便再遇到时可直接引用而不必重新求解,因而可解决所对应的子问题树中的子问题呈现大量重复的问题。

3　解决背包问题的DP算法

3.1　求解过程描述

假定 $W>0$,且两个正整数数组分别为:$(v_1,v_2,\cdots,v_n)$ 和 $(w_1,w_2,\cdots,w_n)$,期望找到 $\sum_{i=1}^{n}v_ix_i$ 最大且 $\sum_{i=1}^{n}w_ix_i\leqslant W$ 的子集为 $T\in\{1,2,\cdots,n\}$。采用上述动态规划思想,解决背包问题的求解过程可描述如下:

步骤一:将问题分解为小的子问题。

构造一个二维数组 $V[0\cdots n,0\cdots W]$

$1\leqslant i\leqslant n$ 且 $0\leqslant w\leqslant W$

$V[i,w]$ 用来存储权值至少为 w 且值最大的子集 $T\in\{1,2,\cdots,n\}$

即:$V[i,w]=\max\{\sum_{j\in T}v_j:T\in\{1,2,\cdots,i\},\sum_{j\in T}w_j\leqslant w\}$

计算出所有满足条件的此二维数组的值,则 $V[n,W]$ 就包含解决问题的解决方案。

步骤二:根据子问题的解递归定义最优解的值。

设置初始状态集合:

$V[0,w]=0$ 当 $0\leqslant w\leqslant W$,没有任何项目时

$V[i,w]=-\infty$ 当 $w<0$,　　非法

递归步:

$V[i,w]=\max(V[i-1,w],v_i+(V[i-1,w-w_i])$,其中 $1\leqslant i\leqslant n,0\leqslant w\leqslant W$[3]。

式中:$V[i-1,w]$ 为上一轮中重量为 w 时所求得的价值最大值;$v_i+(V[i-1,w-w_i]$ 表示:选取 w_i 时的价值为 v_i,此时允许的最大的重量为 w,$w-w_i$ 是还剩下的可承重量。

$V[i,w]$ 的取值为 $V[i-1,w]$ 和 $v_i+(V[i-1,w-w_i]$ 中的较大值。如果有 i 种物品可以选择,而此时所能承受的最大重量为 w 时,只有两种选择:

① 不选新加入的第 i 种物品。此时只有 $i-1$ 种物品,最大承受重量为 w 的最大价值为 $V[i-1,w]$。

② 选择新加入的第 i 种物品。第 i 种物品的重量为 w_i 的价值为 v_i,现所能承受的最大重量为 $w-w_i$;而 $i-1$ 种物品还可以选择,此时的最大价值为 $V[i-1,w-w_i]$;因此价值最大值为$(v_i+V[i-1,w-w_i])$。

步骤三:自底向上进行计算 $V[i,w]$。

底:$V[0,w]=0$　当 $0\leqslant w\leqslant W$;

自底向上计算:使用公式 $V[i,w]=\max(V[i-1,w],v_i+(V[i-1,w-w_i])$。

3.2　实现算法

```
KnapSack(v,w,n,W)
{
for (w=0 to W) V[0,w]=0;
//将二维数组第一行赋值全零
for (i=1 to n)
  for (w=0 to W)
    if (w_i≤w)
      V[i,w]=max(V[i-1,w],v_i+(V[i-1,w-w_i])
// V[i,w]纪录权值至少为 w 且值最大的子集{1,2,…,n}
    else
      V[i,w]=V[i-1,w];
Return V[i,W];
}
```

另外再定义一数组 keep[i,K]纪录所选择的物品。当第 i 个物品选中时,将其赋值为1,否则为0。

```
K=W
for (i=n downto 1)
  If (keep[i,K]==1)
    {
      output i;
      K=K-w[i]
    }
```

输出 i 的值即为所选择物品对应的编号。

3.3　测试结果

在Matlab6.5[4]环境下对该算法进行测试的过程和结果如下:

```
>> v=[10 40 30 50];w=[5 4 6 3]; n=4;W=10;
>> [V,out]=KnapSack(v,w,n,W)
V=
   0 0 0 0 10 10 10 10 10 10
   0 0 0 40 40 40 40 40 50 50
   0 0 0 0 0 40 40 40 50 70
   0 0 50 50 50 50 50 50 90 90
out=0 2 0 4
```

数组 out 输出的 i 值表示物品的编号。结果表明选择第二和第四种物品,所得到的最大价值为90。此算法的时间复杂度为 $O(nW)$[5]。

4　与其他方法的比较与分析

一般来说,用来解决背包问题的方法主要有递归法和贪心法等,但用这两种方法来解决背包问题都有其不可避免的缺点。递归法虽能遍历搜索空间,找到最优解,但由于此问题的解的空间是以级增长的,所以它只适用于解决小规模的背包问题;而贪心法又很难真正找到最优解,即使能找出较优的解也往往与真正的最优解相差很远。如果采用自顶向下的分治法解决此问题,将会随着物品的增多以级规模指数递增,因为分治法要重复计算相同的值。但采用动态规划算法,则可解决重复计算相同内容的问题,以达到减少冗余的根本目的。由于采用的动态规划实质上是一种以空间换时间的技术方法,因此在实现过程

(下转第92页)

1)持球队员决策。

对于持球队员来说,首先考虑的是把球控制在自己的范围内,如果球被对方球员截到,进攻就无从谈起。在对手离球较近(其威胁性也大)的情况下,持球队员优先执行各种控球的策略,对球进行彻底的控制。而在对手离球较远(其威胁性也小)的情况下,持球队员才可以考虑如何组织进攻。组织进攻包括自己是否应该继续带球、预测周边的队友下一步的动作(以便更好地传球)及其传球的路线选择等。持球队员的决策在一个完整的进攻中处于核心地位(可以直接影响球的位置和速度),同时可以对无球队员决策进行方向性的指导。好的持球队员的决策决定了进攻的成败。持球队员是进攻的发起者。

2)无球队员决策。

对于无球队员来说,无须考虑控球的问题,也暂时无法影响球的位置和速度。无球队员的决策是以对持球队员行为的预测为核心。无球队员根据预测出持球队员动作(带球或传球)的情况,决定自己的行为。无球队员的行为主要是跑位,根据设定的有利进攻点来调整自身的方向。同时无球队员应做好对方可能会断球、球被断后进行快速抢球及其完成从无球队员到持球队员转换的准备。无球队员是进攻的响应者。

如图 1(a)所示,持球队员首先预测了队友的动作(图中虚线箭头所示),将球沿实线箭头传给队友;同时如图 1(b)所示,无球队员首先预测了队友的动作(传球),并预测球的路线(图中虚线箭头所示),同时决定沿实线箭头跑位。比较同一情况下两名不同位置的球员的决策,可以发现虽然预测和实际的动作有出入,但从总体上可以认为这两个 agent 完成了配合,达到了想要的结果(实现了非通讯状态下的球员之间的合作,即球员之间产生了"默契")。

实验表明,这种以持球队员为核心的进攻策略简化了 agent 考虑的因素,提高了传球的目的性,加快了进攻的速度,从而实现了预期的球员之间的"默契"。

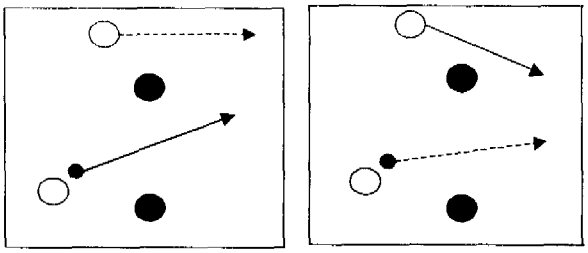

图 1 同一情况下球员的决策

4 结束语

从多 agent 系统研究入手,根据 RoboCup 的具体情况,实现了非通讯状态下 agent 之间的配合。在调整了考虑的 agent 的数目后,我们的策略达到了预期的效果。下一步,考虑增加对对手的建模,提高传接球的质量,以期得到更好的进攻成功率。

参考文献:

[1] 咸鹤群,孟庆春,殷 波,等.多 Agent 系统中潜在角色值研究[J].哈尔滨工业大学学报,2003,35(9):1089-1092.

[2] 祁正华,任勋益.基于 MAS 的智能决策支持系统研究[J].微机发展,2004,14(9):14-16.

[3] 陈进才.多 Agent 系统的形式化理论研究[D].西安:西安交通大学,2000.

[4] Kok J R,Spaan M T J,Vlassis N. Non-communicative multi-robot coordination in dynamic environments[J]. Robotics and Autonomous Systems,2005,50:99-114.

[5] 张颖霞,杨宜民,陈 波,等.多智能体团队合作在机器人足球赛中的应用[J].微机发展,2004,14(7):112-114.

(上接第 89 页)

中,不得不存储实施过程中产生的各种状态,而造成空间复杂度比其它算法要高。

文中选择动态规划算法来解决背包问题,且认为它在空间上是可以承受的,而采用搜索算法在时间上却是难以承受的。舍空间而取时间,这正是该算法在解决背包问题上体现出的一种优越性。

5 结束语

文中把动态规划方法用于求解背包问题,其算法是以空间换时间来解决问题的一种有效的方法。在解决实际问题的过程中,面对复杂问题而分解出的子问题往往并不是孤立的,要解决问题的子问题常常需要调用相同的子问题的子问题。如果它们之间也不是独立的,采用分治法势必会在解决同样的问题上造成很大的时间浪费。由于动态规划方法对一个问题或多次出现的相同问题仅需解决一次,因此可节省大量的计算时间而具有更高的运行效率。

参考文献:

[1] 玄光男,程润伟.遗传算法与工程优化[M].北京:清华大学出版社,2004.55-63.

[2] 罗小虎,赵 雷.一个解决 0/1 背包问题的蚁群算法[J].苏州大学学报(工科版),2004(2):41-44.

[3] Supowit K. Finding a Maximum Planar Subset of a Set of Nets in a Channel[J]. IEEE Transations on Computer-Aided Design of Integrated Circuits and systems,1987,6(1):93-94.

[4] 阮沈勇,王永利,桑群芳.Matlab 程序设计[M].北京:电子工业出版社,2004.

[5] 施益昌,郑贤斌,李自立.基于 Matlab 动态规划中最短路线的实现程序[J].电脑学习,2003(12):38-40.