doi:10.11887/j.cn.202203005 http://journal.nudt.edu.cn

面向多最优解组合优化问题的决策求解算法*

胡振震,袁唯淋,罗俊仁,邹明我,陈 璟
(国防科技大学 智能科学学院,湖南 长沙 410073)

摘 要:针对具有固定物品总和、多最优解特征的组合优化问题,以固定总和实数子集问题和购买鸡翅问题为例,给出了这类多最优解组合优化问题的形式化表示。在分析枚举等经典算法基础上,提出了基于整数状态表示和实数状态表示的0-1决策递归搜索多最优解动态规划算法。针对该算法在最优解数量较大时,时间复杂度趋向$O(m^n)$的问题,提出了基于相同决策路径合并和基于0-x决策的两种改进算法。实验中两种改进算法的计算时间基本符合与$O(nb+nm)$的正比关系,表明对于这类多最优解组合优化问题具有良好的求解性能。

关键词:组合优化;多最优解;动态规划;固定总和实数子集问题
中图分类号:O221,O158 **文献标志码**:A **开放科学(资源服务)标识码(OSID)**:
文章编号:1001-2486(2022)03-031-10

Decision solving algorithm for multiple optimal solution combinatorial optimization problem

HU Zhenzhen, YUAN Weilin, LUO Junren, ZOU Mingwo, CHEN Jing
(College of Intelligence Science and Technology, National University of Defense Technology, Changsha 410073, China)

Abstract: Oriented to the combinatorial optimization problem with fixed sum of goods and multiple optimal solutions, the problem formulation was given by two examples: the fixed sum real number subset problem and buying wings problem. A integer state and a real number state multi-optimal solution dynamic programming algorithm based on 0-1 decision recursive search was put forward on the foundation of analysis of some classical methods like enumeration. In order to cope with the problem of time complexity tending to the extreme $O(m^n)$ when the number of optimal solutions is large for the proposed algorithms, two improved algorithms, the same decision path fusion algorithm and the 0-x decision based algorithm were proposed. The computation time of the improved algorithms is consistent with the proportional relation with $O(nb+nm)$ on the whole in experiments, which indicates that these algorithms have good performance for this type of problem.

Keywords: combinatorial optimization; multiple optimal solution; dynamic programming; fixed sum real number subset problem

最优化是应用很广的一个数学分支,很多问题都可以看成是某种形式的最优化问题。最优化问题根据变量是否连续分为两类,其中离散变量的优化问题称为组合优化问题。组合优化通常是在一定的约束条件下,根据目标函数要求,从一个有限的集合里寻找一个最优的对象,如一个数、一个集合,或者一个排列[1-3]。在一定约束条件下求目标函数的极值问题也称为数学规划,因此组合优化问题也是规划问题。组合优化的很多理论也基于规划,如线性规划、动态规划、整数规划等[4-6]。

组合优化的概念源于计算科学,与运筹、决策、管理、经济等学科紧密相关,科学研究和工程应用十分广泛。当前组合优化已经发展成一个涵盖许多规划问题的学科,比如:旅行商问题、背包问题、划分问题、指派问题、最短路径问题、网络最大流问题、最小费用流问题、最小顶点覆盖问题、最小支配集问题、最小生成树问题、斯坦纳最小树问题等。尽管组合优化研究已经取得了丰硕的成果,但因为现实问题的复杂性,仍然还有很多问题需要研究:一是针对新应用问题的模型和方法研究,比如线性约束下的平行机排序问题[7]、“新高改”高中“走班”排课问题[8]、连续背包问题[9]、矩形背包问题[10]等;二是针对已有问题的新方法研究,比如寻路问题的规范排序A星算法[11]、图匹配问题的深

* **收稿日期**:2021-06-03
基金项目:国家自然科学基金资助项目(61702528,61806212);湖南省自然科学基金资助项目(2019JJ50724)
作者简介:胡振震(1984—),男,浙江武义人,博士研究生,E-mail:hzzmail@163.com;
陈璟(通信作者),男,教授,博士,博士生导师,E-mail:chenjing001@vip.sina.com

度强化学习方法[12]、旅行商问题的图卷积网络方法[13]、背包问题的强化学习方法[14]、二次背包问题的剪枝算法[15]等。

本文是针对一类新问题开展研究。现实中很多组合优化问题只有一个最优解,只要找到它就可完成求解,但也存在一些特殊的问题具有多个最优解,而且决策者必须获取全部最优解才能据此适时做出决策,比如财务预算要在预算总值固定情况下得到所有的可能分配方案。这类问题具有两个显著特征:一是最优解是多个且需全部求出;二是目标物品或数值的总和是固定的。两个特征使该类问题有别于单最优解问题,可称之为多最优解组合优化问题。目前专门针对多最优解组合优化问题的研究少有报道,本文将其作为研究对象并以固定总和实数子集问题和费城中国餐馆购买鸡翅问题为例开展建模和分析。

1 相关工作

组合优化是离散变量的最优化问题,往往可以写成线性规划的形式。一个求最小目标的问题可用如下数学模型描述:

$$\min F(\boldsymbol{x}) \quad \text{s.t.} \begin{cases} G(\boldsymbol{x}) \leqslant 0 \\ \boldsymbol{x} = (x_1, x_2, \cdots, x_i)^{\mathrm{T}} \\ x_i \in D \end{cases} \tag{1}$$

其中,$\boldsymbol{x}$ 为决策变量,D 表示问题或者决策变量的定义域(有限个值组成的集合),F 为目标函数,$G(\boldsymbol{x}) \leqslant 0$ 为约束条件。当求解的变量是整数时,转变为整数规划问题。

线性规划问题常用的求解算法主要是单纯形法及其相关扩展算法。整数规划中的两种主要算法——分枝限界法和割平面法与线性规划的算法具有密切联系[5]。对于变量取值是0或1的整数规划问题(称为0-1整数规划问题),有两种更为直观的求解方法:一是枚举法,即枚举所有变量等于0或1的所有组合,然后判断所有组合是否满足约束条件并使目标函数最优;第二种是隐枚举法,同样是枚举所有的组合,但引入一些过滤条件,过滤掉局部不满足约束或目标函数值非优的组合,从而减少计算量[16]。0-1整数规划问题,也可以看作是多阶段决策问题,从这一角度建模可以运用动态规划的方法求解。动态规划不是一种特殊的算法,而是一种考察问题的途径和思路。针对不同的问题,往往需要建立针对性的模型进行求解[5,17]。

除了确定性算法外,组合优化问题还可采用启发式算法。无论是人为的构造启发式,还是基于一些最优化方法(如禁忌搜索、模拟退火、遗传算法等)设计启发式,主要目的是解决大规模问题中确定性算法无法在有限时间内求得精确解的问题,即利用启发式算法在有限时间内获得可接受的解(如近似解)。启发式往往依赖于问题的性质,需要针对不同类型的问题修改,所以通常情况下启发式设计并不容易。近年来深度强化学习被发现具有一定的潜力用来求解组合优化问题,可以通过学习来获得好的启发式[18-22]。但这些方法主要是针对单最优问题。从需要获得多最优解的角度看,上述方法中枚举法、隐枚举法、基于动态规划原理的算法最具有获得问题的全部解的潜力,所以本文的研究主要集中于此。

2 问题定义与求解

2.1 问题形式化描述

多最优解组合优化问题可以定义为:从一个有限集合中寻找多个对象、数、集合或排列,使目标达到最优。它要求将多个最优解全部求出,且约束为等式。在数学形式上可以表示为:

$$\text{Find all } \boldsymbol{x} = \arg(\min F(\boldsymbol{x})) \quad \text{s.t.} \begin{cases} G(\boldsymbol{x}) = 0 \\ \boldsymbol{x} = (x_1, x_2, \cdots, x_i)^{\mathrm{T}} \\ x_i \in D \end{cases} \tag{2}$$

固定总和实数子集问题和费城中国餐馆购买鸡翅问题是两个典型示例。固定总和实数子集问题源于实际预算问题:数据集 X 包含一系列数据{8.05, 6.98, 6.19, 5, 22.96, 4.71, 4.74, 4.25, 6.34, 2.77, 7.31, 3.59, 18.28, 19.55},希望从中找到多组数(多个子集)使其累加总数为84.03。如果把问题的求解变量看作是取值为{0,1}的变量,即把问题看作是在数据集 X 中对每个数据做挑选的决策,选中则变量取为1,不选中则取为0,如式(3)所示:

$$\begin{cases} \text{Find all } \boldsymbol{x} = \arg(\boldsymbol{c}\boldsymbol{x} = 84.03) \\ \boldsymbol{c} = (8.05, 6.98, 6.19, 5, 22.96, 4.71, 4.74, \\ 4.25, 6.34, 2.77, 7.31, 3.59, 18.28, 19.55) \\ \boldsymbol{x} = (x_1, \cdots, x_i, \cdots, x_{14})^{\mathrm{T}} \\ x_i \in \{0,1\} \end{cases} \tag{3}$$

购买鸡翅问题源于一家位于美国费城的中餐馆,其鸡翅菜单与众不同,不标注鸡翅单价而是标注不同数量鸡翅组合的价格,如表1所示。问题

是求解购买指定数量鸡翅时的全部最优惠购买方案。类似地,可以把问题建模为 0 - 1 整数规划问题(0 - 1决策问题),如式(4)所示:

$$\text{Find all } \boldsymbol{x} = \arg(\min(\boldsymbol{vx}))$$

$$\text{s.t.}\begin{cases}\boldsymbol{cx}=b\\ \boldsymbol{c}=(\underbrace{4,\cdots,4}_{b/4},\cdots,\underbrace{200,\cdots,200}_{b/200})\\ \boldsymbol{v}=(\underbrace{4.55,\cdots,4.55}_{b/4},\cdots,\underbrace{222.5,\cdots,222.5}_{b/200})\\ \boldsymbol{x}=(x_1,\cdots,x_i,\cdots,x_n)^{\mathrm{T}}\\ x_i\in\{0,1\}\end{cases} \tag{4}$$

其中,b 为要购买的鸡翅总数。

表 1　鸡翅菜单

Tab. 1　Wings menu

数量	价格	数量	价格	数量	价格
4	4.55	18	20.40	35	39.15
5	5.70	19	21.55	40	44.80
6	6.80	20	22.70	45	50.50
7	7.95	21	23.80	50	55.60
8	9.10	22	24.95	60	67.00
9	10.20	23	26.10	70	78.30
10	11.35	24	27.25	75	83.45
11	12.50	25	27.80	80	89.10
12	13.60	26	28.95	90	100.45
13	14.75	27	30.10	100	111.25
14	15.90	28	31.20	125	139.00
15	17.00	29	32.35	150	166.85
16	18.15	30	33.50	200	222.50
17	19.30				

严格地说,固定总和实数子集问题是一个多最优解的约束满足问题,但也可以转换为最优化问题;购买鸡翅问题则是一个标准的优化问题,要求购买固定数量鸡翅的同时使得花费的代价最小。

2.2　枚举和隐枚举法

枚举和隐枚举法是多最优解组合优化问题的直观解法。以固定总和实数子集问题为例,引入一对相反的不等式,式(3)的约束问题就能转换为一个优化问题,如式(5)所示。这是典型的整数规划形式,可以理解为 $n=14$ 个变量(阶段)的 0 - 1 整数规划(决策)问题。

$$\text{Find all } \boldsymbol{x} = \arg(\max(\boldsymbol{cx}))$$

$$\text{s.t.}\begin{cases}\boldsymbol{cx}\leqslant b\\ \boldsymbol{c}=(8.05,6.98,6.19,5,22.96,4.71,4.74,\\ 4.25,6.34,2.77,7.31,3.59,18.28,19.55)\\ \boldsymbol{x}=(x_1,\cdots,x_i,\cdots,x_{14})^{\mathrm{T}}\\ x_i\in\{0,1\}\\ b=84.03\end{cases} \tag{5}$$

首先考虑枚举法,由于数据集中总共包含 14 个数据,对应于 14 个变量,因此任意数量(1 ~ 14)的数据都可以进行组合,枚举组合总数为 $2^{14}-1$。枚举过程可以利用字典序循环实现。由于目标函数是所选数据的和,所以每个组合都需要累加计算,运算量接近 14 乘以枚举组合数。因此,对于数据集规模为 n 的问题,枚举方法的时间复杂度约为 $O(n2^n)$;枚举过程不额外增加存储空间,所以空间复杂度为 $O(n)$。显然时间复杂度的指数级增长源自枚举组合数随数据集规模增大产生的指数级增长。

隐枚举法思想有些类似于分支定界法,基本思路是:当检测到某些分支组合不符合约束要求,则避免进一步往下分支,从而减少总的计算量。隐枚举过程可以理解为搜索树的扩展,数据集中的每一个数作为一层的决策,以选择和不选择两种情况扩展分支,因此搜索树是一个不断扩展的二叉树,如图 1 所示。图中“ × ”表示当前节点已经不满足约束条件,因此可以剪掉。如果不考虑剪枝,那么隐枚举法的决策树从根节点到第 14 层的叶子节点是完整的,叶子节点数量为 2^{14},因此时间复杂度为 $O(n2^n)$,与枚举法相同。若考虑剪枝则可以使其计算复杂度与枚举法相比有明显的下降,但下降的程度与问题的求解特征和方式相关,也与剪枝约束条件相关,不是一个固定值。

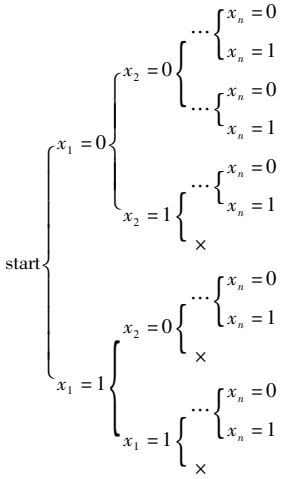


图 1　隐枚举法搜索树

Fig. 1　Searching tree of implicit enumeration method

2.3 基于动态规划的方法

若将问题看作多阶段决策问题，可以自然地运用动态规划原理来建模求解。对于一个 n 阶段的决策问题 $s_{k+1}=f(s_k,x_k)$，$k=1,\cdots,n$，其中 s_k 是第 k 阶段的系统状态，x_k 是第 k 阶段的决策变量，优化问题可以看作求决策向量 $x_1,x_2,\cdots,x_n$ 使得目标函数（指标）达到极值：

$$\max f = \sum_{k=1}^{n} V_k(s_k,x_k) \tag{6}$$

式中，V_k 为各阶段的目标函数。根据动态规划的最优性定理，可以将求指标极值问题转换为分阶段的递推过程：

$$\begin{cases} \max f_k = \max(V_k(s_k,x_k)+f_{k+1}) \\ f_{k+1} = \sum_{i=k+1}^{n} V_i(s_i,x_i) \end{cases} \tag{7}$$

或者：

$$\begin{cases} \max f_k = \max(V_k(s_k,x_k)+f_{k-1}) \\ f_{k-1} = \sum_{i=1}^{k-1} V_i(s_i,x_i) \end{cases} \tag{8}$$

式(7)和式(8)分别称为逆序和顺序的动态规划方程[4]。由此多阶段的决策问题可以分解为在各个阶段的状态和目标函数递推过程中做出最优的决策。

为解决固定总和实数子集问题，首先用一个简单的示例来考察动态规划的求解结构。问题是从数据集[1,2,3,4]中选择数据子集使其和为7。这是一个完全类似的问题，可表示为整数规划的形式，并转换为多阶段决策问题：

$$\text{Find all } \boldsymbol{x} = \arg(\max f)$$
$$\text{s.t.} \begin{cases} f = x_1+2x_2+3x_3+4x_4 \leqslant 7 \\ \boldsymbol{x} = (x_1,x_2,x_3,x_4)^{\mathrm{T}} \\ x_1,x_2,x_3,x_4 \in \{0,1\} \end{cases} \tag{9}$$

先考虑顺序递推的方法，定义状态为决策（选择该阶段决策变量数据）前的数据总和。实际状态的转移和指标的递推为：

$$\begin{cases} s_1=0,\ f_1=0 \\ s_2=s_1+x_1,\ f_2=\max(x_1+f_1) \\ s_3=s_2+2x_2,\ f_3=\max(2x_2+f_2) \\ s_4=s_3+3x_3,\ f_4=\max(3x_3+f_3) \\ s_5=s_4+4x_4=7,\ f_5=\max(4x_4+f_4) \end{cases} \tag{10}$$

根据各阶段的状态转移式(10)可知，求决策变量需要从最后一个阶段开始，若 f_4 的所有可能值已知，则使 f_5 最优可以将 x_4 求出，进而知道 f_4 的最优值，进一步将 x_3 求出来，不断递推直到求出所有阶段的决策变量。对于这样一个求解过程，可以考虑两种结构：一种是循环，一种是递归。循环的结构是先将 $k-1$ 阶段的所有状态 s 和指标 f 求出，再求 k 阶段的状态和指标，直到最后一个阶段，然后再求解决策变量。通俗讲就是向前建表（递推状态和指标），向后查表（求决策变量）。而递归的结构是以递归的方式从最后一个阶段的状态开始递归，当存在前一阶段的未知状态和指标时，则递归地进入前一个阶段的计算中，直至第一个阶段的状态和指标计算完成，这样等价于完成了建表的过程，接着通过查表可以求解决策变量。

图2给出了顺序递推形式下利用循环方法构造的求解结构。图中的曲线表示递推的过程。从状态 s_1 和指标 f_1 开始递推到 s_5 和 f_5，根据式(10)可知 $f_5=7$ 为最优的目标。根据递推的过程可以计算得到 $x_4=1$，$s_4=3$，根据 $s_4=3$ 状态可以知道有两条递推路径到它。如果是只有一个解，那么查询过程只要找到一条解路径即可，但对于多最优解问题，必须获取全部路径，可以发现 $x_4=1,x_3=1,x_2=0,x_1=0$ 和 $x_4=1,x_3=0,x_2=1,x_1=1$ 是两个最优解，图中两条蓝色的路径就是最优解的路径。

图3给出了顺序递推形式下利用递归方法构造的求解结构。图中的曲线表示递归调用时的过程。根据约束条件状态 $s_5=7$ 已知，但 f_5 是未知

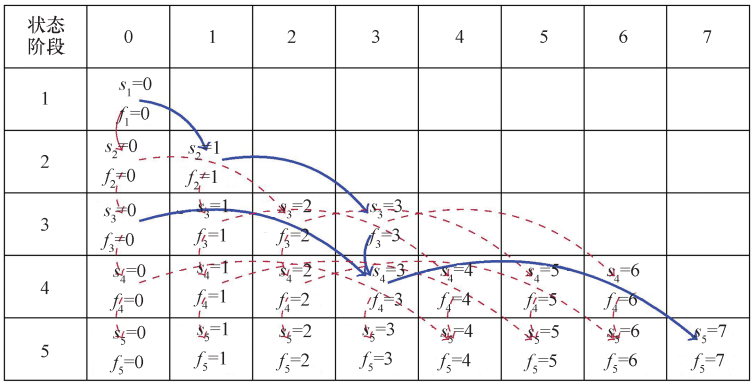


图2 顺序递推循环求解结构

Fig. 2 Loop solving structure for sequential recursion

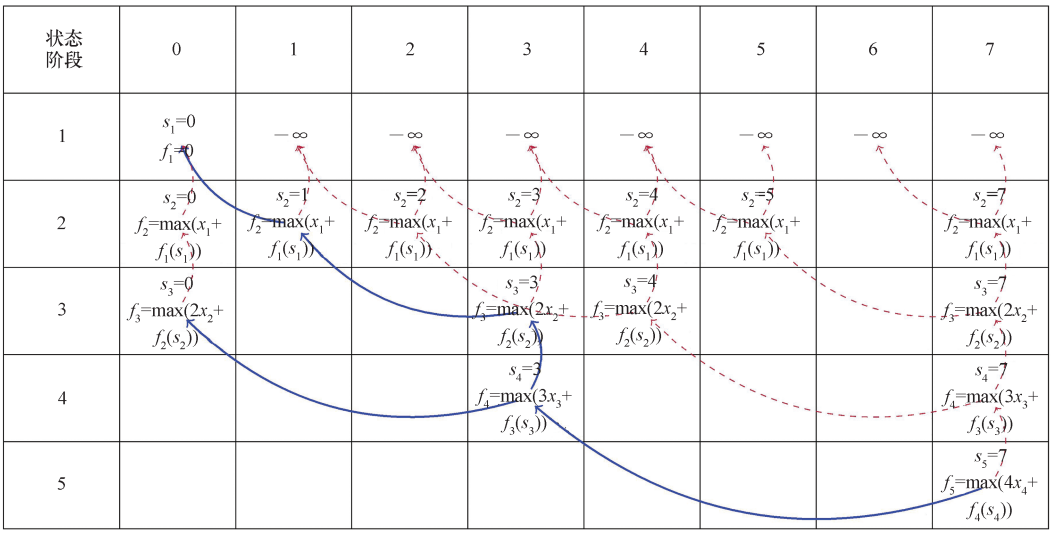


图 3　顺序递推递归求解结构

Fig. 3　Recursive solving structure for sequential recursion

的,要求 f_5 则需递归的求 f_4,接着 f_3,直到 f_1。因为状态 s 定义为决策前的和,因此 f_1 只有一个值为 0,其他状态下的 f 值都定义为负无穷。那么递归完毕同样得到所需的 f 值后,也可以根据查表得到解。

从上述分析可知,顺序递推形式下,通过循环或者递归都能得到解。类似地,逆序递推形式同样也可以使用循环和递归两种求解结构,因此一个动态规划问题求解在实现上可以有四种结构,而且可以用图证明逆序循环的结构和顺序递归的结构是一致的,而逆序递归和顺序循环的结构是一致的。总的来说,顺序和逆序形式与指标推导式相关,顺序递推是指标从前往后推,决策变量从后往前计算;逆序递推是指标从后往前推,决策变量从前往后计算。循环和递归则与指标递推式的具体实现相关,循环是先计算指标推导式的等号右侧项,而递归是先调用指标推导式的等号左侧项。另外状态的定义也不是固定的,也可以定义为各阶段决策完成后的状态,对应的则需修改递推式的下标形式。

2.4　基于 0 -1 决策动态规划多最优解算法

根据前述分析,在动态规划递推建表过程中,多个最优解能够用递推路径表示,与单最优解问题的差异主要体现在查表求决策变量过程中。一般的简单递推式查表,只能得到一条最优解路径,但是多最优解问题必须得到多条最优解路径。

以顺序递推循环求解结构为例,单个最优解问题可以用一个很简单的逻辑进行求解:从最后一个阶段开始,最优的 f_n 已知,如果 f_n ! $=f_{n-1}$,那么必有 $x_n=1$,如果 $f_n==f_{n-1}$,那么必有 $x_n=0$,然后根据最优的 f_{n-1} 在前一个阶段继续求解。但若要得到多条解路径则不同。

一种直观解决思路是考虑建表过程中,当递推某阶段出现多种不同决策可以得到相同指标时,记录多个决策,并在查表时根据这一信息向后搜索。然而这种方式会带来内存空间的增大,因此若不希望增大内存空间,那么只能在建表的信息里实现对路径搜索。观察图 2 中第 3 和第 4 阶段,目标状态 $f_4=3$ 可以从 $f_3=0$ 和 $f_3=3$ 得到,这是两条路径。不同于单最优解的逻辑,在查表过程中已知最优 f_n 情况下,应判断 $f_n==f_{n-1}+c_{n-1}x_n$ 和 $f_n==f_{n-1}$ 两种情况,当都满足条件时则采用分支的形式继续下一阶段。由于已经在建表过程中确认解的存在,因此完全可以采用深度优先搜索的方式得到不同路径的解,而且搜索分支不用深入第 1 阶段,只要达到 $f_n==0$(状态也等于 0)就可以终止分支,如算法 1 所示。

算法 1　整数状态的多最优解动态规划算法

Alg. 1　Integer state multi-optimal solution dynamic programming algorithm

```
变量:c 代价,v 价值,b 边界,p 阶段,s 状态,f 指标函数,res 结果
for p = 1 to n do:                          #建表
  for s = 0 to b do:
    f[p][s] = max(v[p-1]+f[p-1][s-c[p-1]],f[p-1][s])
function findX(p,s,res):                    #查表
  if s==0: return res
  if f[p][s]==f[p-1][s-c[p-1]]+v[p-1]:
    nres = copy(res); nres.append(p)
    findX(p-1,s-c[p-1],nres)
  if f[p][s]==f[p-1][s]: findX(p-1,s,res)
```

分析算法的结构可知,在状态是整数的情况下,利用循环结构来实现状态递推,可以方便地利用数组(或列表)数据结构来记录指标(目标函数)信息,并可同时利用数组的索引信息来表示阶段和状态。但是对于如式(3)这样的实数问题,阶段是整数可方便表示,但状态却是实数的,不便于表示。因此基于动态规划的基本原理,考虑顺序递推的循环结构,针对性地提出一种状态实数表示的算法,即采用列表加字典的数据结构算法,核心是利用实数表示的状态作为字典的 key 值,如算法 2 所示。

算法 2　实数状态的多最优解动态规划算法

Alg. 2　Real number state multi-optimal solution dynamic programming algorithm

变量: c 代价,v 价值,b 边界,p 阶段,s 状态,f 指标函数,*res* 结果,x 决策变量,D 为 x 的值域

```
f = [{0.0:0.0}]                          #元素为字典的列表
for p = 1 to n do:                       #建表
  f.append({})                           #加入空字典
  for i = 1 to len(f[p-1]) do:
    s = list(f[p-1].keys)[i]
    for x in D do:
      fv = f[p-1][s] + v[p-1]x; s = s + c[p-1]x
      if s≤b && s∉f[p]: f[p][s] = fv
      if s≤b && s∈f[p]: f[p][s] = max(fv,f[p][s])
function findX(p,s,res):                 #查表
  if s = =0: return res
  if f[p][s] - f[p-1][s - c[p-1]] + v[p-1] < ε:
    nres = copy(res); nres.append(p)
    findX(p-1,s - c[p-1],nres)
  if f[p][s] - f[p-1][s] < ε: findX(p-1,s,res)
```

实数状态的多最优解算法与整数状态的算法基本结构一致,差异主要由使用实数作为字典的 key 而产生。要注意由于实数在计算机中只是高精度的近似表示,所以在作为 key 以及运算时可能产生的问题可以使用固定模式的实数表达来解决。当然如果从另外一个角度看,实数状态的问题也可以直接转换为整数状态的问题来解决,比如:式(5)可以将 c 和 b 同乘以 100,那么整个问题变为整数问题,而且解是相同的。分析算法 1 可以知道,基于 0 - 1 决策的动态规划法的空间复杂度主要由指标 f 的矩阵决定,为 $O(nb)$,b 为整数表示的状态总数(对于上述实数问题,$b = 84.03 \times 100 = 8\ 403$)。对于时间复杂度,计算主要体现在建表和查表过程中,建表过程中计算的区域约为 $nb/2$,加上一些比较运算,可以认为在 $O(nb)$ 量级。而查表过程由解的数量 m 和表的阶段数 n 决定,最小的情况是 nm,而最极端的情况是 m^n,因此时间复杂度在 $O(nb + nm)$ 和 $O(nb + m^n)$ 之间的范围内。

比较算法 2 与隐枚举算法,建表过程中的 $s \leqslant b$ 判断可以看作隐枚举法的剪枝约束,而动态规划相比隐枚举法的主要优势在于建表过程中对状态的规约,相同的状态合并到一起考虑,而隐枚举法一直都是以叉树的形式在扩展。基于动态规划的算法正是在对状态的不断规约中使得其分支数不断地缩减从而减小计算量。

3　两种改进的多最优解算法

前面提出的基于 0 - 1 决策的算法有效解决了本文多最优解问题的两个特征带来的问题。固定物品(数值)总和的问题通过固定目标终状态解决,多最优解的问题通过考虑多条解路径查找来解决。同时由于状态的压缩使其相比枚举和隐枚举算法在计算复杂度上具有明显的优势。

购买鸡翅问题,要求购买目标数量的物品,求代价最小的多个最优购买方案,仍采用前述方法开展分析。最直观的求解思路仍然是采用枚举,枚举所有的购买方案组合,从中得到全部的最优惠方案。枚举时需要考虑不同数量组合的鸡翅可以购买多次的问题,比如要购买 12 只鸡翅,可以购买 3 次 4 只,或 2 次 6 只,或 1 次 4 只 1 次 8 只等。因此购买 $b(>200)$ 只鸡翅,需要在 $b/4 + b/5 + \cdots + b/200 = n$ 个数据的组合中寻找,比如购买 256 只鸡翅,需要在 587 个数据中寻找最优组合,那么枚举总数为 $2^{587} - 1$,显然这种规模已无法有效求解。可考虑用隐枚举法和动态规划法,即将问题转换为 n 个阶段的 0 - 1 决策问题。然而,隐枚举法在购买数量上升到一定的程度后计算时间也变得不可接受,图 4 表明采用隐枚举法在购买数量为 28 时计算时间已经呈现指数级上升。

基于 0 - 1 决策的多最优解动态规划算法结果如图 4 中红色虚线所示,图 4 中蓝色实线是根据量级 $O(nb)$ 估计的计算时间。结果表明在多数情况下计算时间会接近正比于 $O(nb)$,但在某些特殊情况下,比如图中购买 120 和 216 只鸡翅的情况,计算时间有跳跃式的上升,而这正是最优解数量较多的情况。由于图 4 只是为说明问题,所以只选取了有限情况(曲线上星号标记)进行计算。

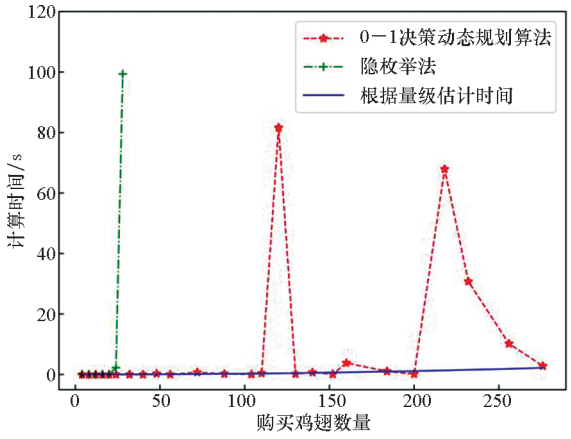


图 4　最优解数量较少情况下的计算时间

Fig. 4　Computation time of cases with small number of optimal solutions

3.1　基于相同决策路径合并的改进算法

显然前述算法在最优解数量较多的特殊情况下,时间复杂度可能会往差的 m^n 方向发展。这是因为当解的数量很多且阶段数量很大时,使用递归的查表可能会近似于搜索一个 n 层 m 叉树问题,这显然是非常耗时的,例如:购买 188 只鸡翅时,最优解的数量为 69,利用前述算法计算时间为 67.1 s;购买 192 只鸡翅时,最优解数量为 300,计算时间已达 2 874.7 s,将变得不可接受。

显然对于要求出全部最优解的给定问题,解的数量 m 是固定的,那么只能从问题的决策阶段上来考虑降低极端条件下的计算复杂性。在前面的问题建模中,采用了 0 - 1 决策思路,其中一些阶段的决策是购买相同数量的鸡翅组合,因此这些阶段的作用是相同的。如果能够对其进行处理和简化,即将相同作用的决策路径合并,那么搜索空间将会大幅度减小。基于这种相同决策路径合并的改进思路,可采用宽度优先搜索实现,如算法 3 所示。

算法 3 核心改进在于宽度优先搜索节点扩展过程中对相同决策效果的节点进行了删减。购买相同鸡翅数量情况下的改进算法与原算法计算时间比较如图 5 所示,可以看到改进算法相比原来算法具有明显的改进,在最优解数量比较多的情况下也没有出现与原算法类似的跳跃性增长。

图 6 给出了购买 4 ~ 300 只鸡翅问题的改进算法计算时间结果。很明显看到,计算时间与最优解数量的关系,计算时间总体符合 $O(nb + nm)$ 的正比估计(图中蓝色线为根据该量级估计的计算时间),但随着解数量的增加有一定的偏离,且解的数量越多则偏离越大,但总体性能仍可接受。

算法 3　相同决策路径合并的改进算法

Alg. 3　Improved algorithm with same decision path fusion

输入: c 代价,v 价值,b 边界,p 阶段,s 状态,f 指标函数

输出: seq 各个最优解(决策变量序列)

```
minf = f[-1][-1], roots = [], seq = []
for p = n to 0 do:
  for s = 0 to b do:
    if s + c[p] = = b && f[p][s] + v[p] = = minf:
      roots. append([p,s,f[p][s],[c[p]]])
for root in roots do:
  opened = [root]
  while opened do:
    node = opened. pop()
    if node[1] = 0&& node[3] ∉ seq:seq. append
(node[3])
    if node[1] ≠0:
      for p = node[0] to 0 do:
        for s = 0 to node[1] do:
          if s + c[p] = node[1] & & f[p][s] + v[p] =
node[2]:
            newnd = copy(node[3])
            newnd. append(c[p])
            if newpath ∉ opened:
              opened. append([p,s,f[p][s],newnd])
```

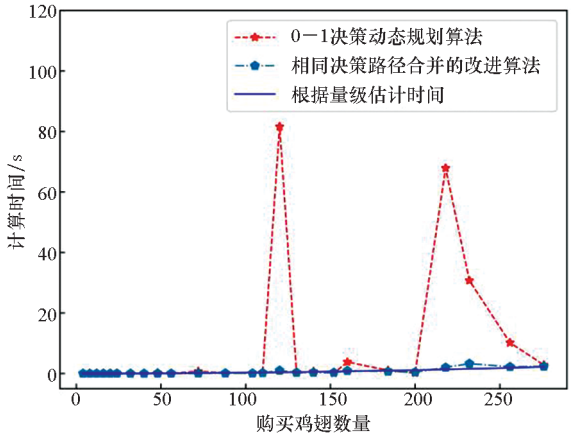


图 5　改进算法与原始算法的计算时间比较

Fig. 5　Computation time comparison between improved algorithm and original algorithm

购买 188 只鸡翅时,改进算法的时间为 2.3 s,是原算法的 1/29;购买 192 只鸡翅时改进算法计算时间为 5.3 s,是原算法的 1/542。

3.2　基于 0 - x 决策的改进算法

对原算法的改进除了进行相同决策路径合并的思路外,其实也可以从建模的角度进行改进。如果能够将 0 - 1 决策问题转变为 0 - x 决策问题,那么决策阶段数也可大幅下降,从而降低极端

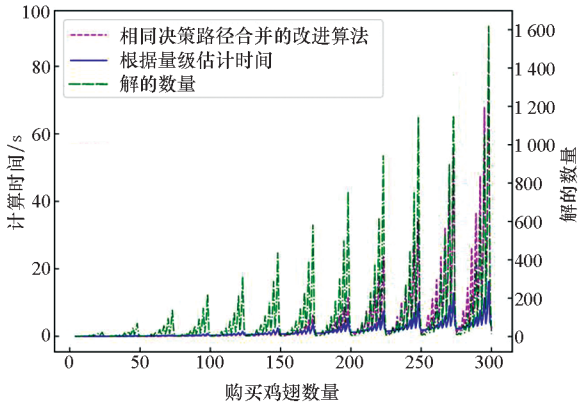


图 6 改进算法计算时间与解数量的关系

Fig. 6 Relation of computation time to solution number for improved algorithm

情况下的时间复杂度。因此考虑将问题建模为 $[0,b/c_j]$ 决策问题,其中每个阶段求解变量的最大值为 b/c_j,数学模型如式(11)所示。

$$\text{Find all } \boldsymbol{x}=\arg(\min(\boldsymbol{vx}))$$

$$\text{s.t.}\begin{cases}\boldsymbol{cx}=b\\ \boldsymbol{c}=(4,\cdots,200)\\ \boldsymbol{v}=(4.55,\cdots,222.5)\\ \boldsymbol{x}=(x_1,\cdots,x_j,\cdots,x_n)^{\mathrm{T}},x_j\in[0,b/c_j]\end{cases}\tag{11}$$

这种改进思路是利用 0 – x 决策建模减少总的决策阶段数,使得表的整体搜索深度明显下降,可采用深度优先类搜索算法实现,计算时间也将有明显的改善。基于 0 – x 决策的改进方法实现如算法 4 所示。

算法 4 基于 0 – x 决策的改进算法

Alg. 4 Improved algorithm based on 0 – x decision

变量: c 代价,v 价值,b 边界,p 阶段,s 状态,f 指标函数,*res* 结果

for p = 1 to n do: #建表

  for s = 0 to b do:

    $f[p][s]=\min([v[p-1]x+f[p-1][s-c[p-1]]$ **for** x = 0 to $b/c[p-1]$ **if** $s-c[p-1]x\geqslant 0]$)

function findX(p,s,*res*): #查表

  if s = =0: **return** *res*

  for x = 0 to $b/c[p-1]$ do:

    if $s-c[p-1]x\geqslant 0$:

      if $f[p][s]=f[p-1][s-c[p-1]x]+v[p-1]$:

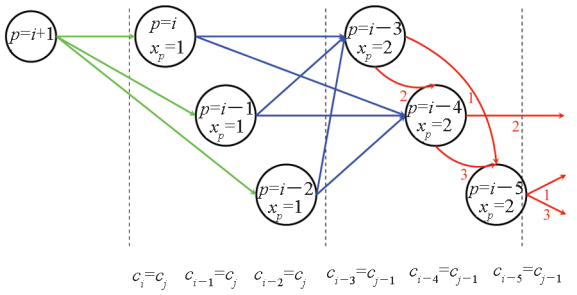
        nres = copy(*res*);

        if $x>0$: *nres*. extend($[c[p-]]x$)

        findX($p-1$,$s-c[p-1]x$,*nres*)

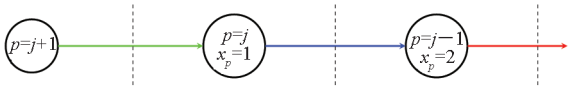
与前述改进算法最大差别是,前述算法采用 0 – 1 决策,而算法 4 采用 $[0,b/c_j]$ 决策,在建表过程中同一阶段的递推需要比较 b/c_j 种选择,在查表过程中的递归则由原来的 2 个分支变为 b/c_j 个分支,原理是通过阶段内更多的比较和计算来换取决策阶段数的下降。

算法 4 的复杂度仍然由建表和查表过程决定,表的结构为 n 行 b 列,只是 n 由原来 0 – 1 决策的 $b/4+b/5+\cdots+b/200$ 变为 $count(4,\cdots,c_j,\cdots,200)$。建表过程的时间复杂度仍为 $O(nb)$,查表过程没有重复分支所以时间复杂度为 $O(nm)$。图 7 给出了 0 – x 决策查表和 0 – 1 决策查表过程的局部差异,其中 j 阶段的 0 – x 决策对应了 i 到 $i-2$ 阶段的 0 – 1 决策。对应于 $x_j=1$,0 – 1 决策有 3 种情况 $x_i=1$ 或者 $x_{i-1}=1$ 或者 $x_{i-2}=1$,而这三种情况其实是一个相同的决策,只是由于 0 – 1 决策结构导致分成 3 个阶段来决策。所以,0 – x 决策的建模避免了 0 – 1 决策中的很多重复搜索,前面基于相同决策路径合并的改进算法实质也是解决这一重复问题。



(a) 0 – 1 决策

(a) 0 – 1 Decision



(b) 0 – x 决策

(b) 0 – x Decision

图 7 0 – 1 决策和 0 – x 决策的查表路径差异

Fig. 7 Searching path difference between 0 – 1 and 0 – x decision

图 8 给出了购买 4 ~ 300 只鸡翅利用基于 0 – x 决策改进算法的运行时间情况,显然计算时间基本符合时间复杂度 $O(nb+nm)$ 的正比估计。与图 6 相比可知,基于 0 – x 决策的改进算法的性能相比基于 0 – 1 决策相同决策路径合并的改进算法也有明显的提升,例如最优解数量最多的购买 298 只鸡翅的情况,计算时间也仅为 0.437 5 s。由此得出结论,对于存在多次相同决策的问题,更

适合建模为 0 – x 决策问题,因为即便采用相同决策路径合并,基于 0 – 1 决策的算法性能也达不到基于 0 – x 决策算法的水平,而每次决策都不相同的问题,则更适合建模成 0 – 1 决策问题。(本文所有实验代码见:https://github.com/hushidong/multi-optimal-solution-combinatorial-optimization-problem)

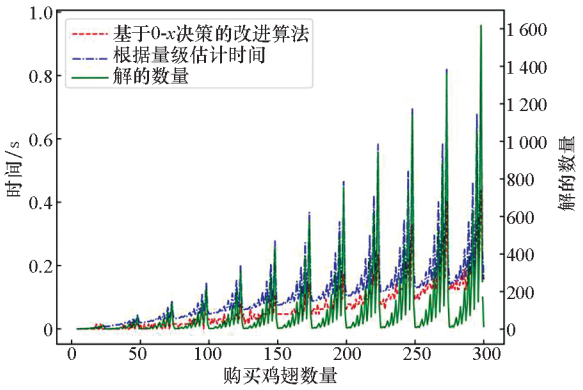


图 8　基于 0 – x 决策改进算法计算时间与解数量的关系

Fig. 8　Relation of computation time to solution number for improved algorithm based on 0 – x decision

4　结论

本文提出了一类具有固定物品(数值)总和及多最优解特征的组合优化问题。该问题不同于一般的单最优解组合优化问题,必须求解所有最优解。以固定总和实数子集问题和费城中国餐馆购买鸡翅问题为例,比较分析了枚举、隐枚举和动态规划三类不同方法。通过动态规划求解结构分析,明确了状态定义、指标递推、决策变量求解对于算法实现的影响。提出基于 0 – 1 决策的多最优解的动态规划算法,并针对实数状态问题提出了基于字典数据结构的实数状态表示求解算法。该算法在最优解数量较少时能够获得较好的性能,但最优解数量较多的情况下,计算时间呈现跳跃式上升。基于降低时间复杂度考虑,提出了压缩决策阶段的改进思路,并实现了两种改进算法:基于相同决策路径合并的 0 – 1 决策求解算法和基于 0 – x 决策的求解算法。结果表明通过减小决策的阶段数可使算法性能得到明显提升,两种改进算法的对比表明具有多次相同决策的问题更适合于建模成 0 – x 决策问题。本文将一类多最优解的特殊的组合优化问题作为一类独立问题专门进行研究,能够帮助解决现实中一些多最优解的决策问题,且对开发新的建模和求解方法也有促进作用,下一步将持续推进算法优化和实际应用。

参考文献(References)

[1]　越民义, 李荣珩. 组合优化导论[M]. 2 版. 北京: 科学出版社, 2014.
YUE M Y, LI R H. Introduction to combinatorial optimization[M]. 2nd ed. Beijing: Science Press, 2014. (in Chinese)

[2]　PAPADIMITRIOU C H, STEIGLITZ K. Combinatorial optimization: algorithms and complexity[M]. New York: Dover Publications, 1998: 1 – 10.

[3]　KORTE B, VYGEN J. Combinatorial optimization: theory and algorithms[M]. 5th ed. Berlin: Springer-Verlag, 2012: 2 – 8.

[4]　董肇君. 系统工程与运筹学[M]. 2 版. 北京: 国防工业出版社, 2007: 297 – 310.
DONG Z J. Systems engineering and operations research[M]. 2nd ed. Beijing: National Defense Industry Press, 2007: 297 – 310. (in Chinese)

[5]　徐玖平, 胡知能. 运筹学[M]. 4 版. 北京: 科学出版社, 2018: 1 – 5.
XU J P, HU Z N. Operations research [M]. 4th ed. Beijing: Science Press, 2018: 1 – 5. (in Chinese)

[6]　LUENBERGER D G, YE Y Y. Linear and nonlinear programming [M]. 3rd ed. New York: Springer, 2009.

[7]　聂嘉明. 线性约束下的组合优化问题研究[D]. 北京: 清华大学, 2017: 17 – 35.
NIE J M. Research on combinatorial optimization problems under linear constraints[D]. Beijing: Tsinghua University, 2017: 17 – 35. (in Chinese)

[8]　郝星星. 组合优化问题的表示方式与进化优化算法研究[D]. 西安: 西安电子科技大学, 2019: 65 – 80.
HAO X X. Research on representations and evolutionary optimization algorithms for combinational optimization problems[D]. Xi'an: Xidian University, 2019: 65 – 80. (in Chinese)

[9]　QUOC H D, TAM N C, NHAN T H N. The continuous knapsack problem with capacities [J]. Journal of the Operations Research Society of China, 2021, 9: 713 – 721.

[10]　SCHULZE B, STIGLMAYR M, PAQUETE L, et al. On the rectangular knapsack problem: approximation of a specific quadratic knapsack problem [J]. Mathematical Methods of Operations Research, 2020, 92: 107 – 132.

[11]　胡书丽. 启发式搜索算法求解组合优化问题的研究[D]. 长春: 东北师范大学, 2019: 24 – 41.
HU S L. The research on the heuristic search algorithms for solving the combinatorial optimization problems [D]. Changchun: Northeast Normal University, 2019: 24 – 41. (in Chinese)

[12]　郭田德, 韩丛英, 唐思琦. 组合优化机器学习方法[M]. 北京: 科学出版社, 2019.
GUO T D, HAN C Y, TANG S Q. Machine learning methods for combinatorial optimization[M]. Beijing: Science Press, 2019. (in Chinese)

[13]　JOSHI C K, LAURENT T, BRESSON X. An efficient graph convolutional network technique for the travelling salesman

problem[EB/OL]. (2019 - 10 - 14) [2021 - 05 - 21]. https://arxiv.org/abs/1906.01227.

[14] BELLO I, PHAM H, LE Q V, et al. Neural combinatorial optimization with reinforcement learning[EB/OL]. (2017 - 01 - 12) [2021 - 05 - 21]. https://arxiv.org/abs/1611.09940.

[15] DJEUMOU FOMENI F, KAPARIS K, LETCHFORD A N. A cut-and-branch algorithm for the quadratic knapsack problem[J]. Discrete Optimization, 2020: 100579.

[16] 周晶. 运筹学[M]. 北京: 机械工业出版社, 2016.
ZHOU J. Operations research[M]. Beijing: China Machine Press, 2016. (in Chinese)

[17] 哈姆迪·塔哈. 运筹学基础[M]. 刘德刚, 朱建明, 韩继业, 译. 10版. 北京: 中国人民大学出版社, 2018.
TAHA H A. Operations research: an introduction[M]. Translated by LIU D G, ZHU J M, HAN J Y. 10th ed. Beijing: China Renmin University Press, 2018. (in Chinese)

[18] AFSHAR R R, ZHANG Y Q, FIRAT M, et al. A state aggregation approach for solving knapsack problem with deep reinforcement learning[C]//Proceedings of Machine Learning Research, 2020, 129: 81 - 96.

[19] BENGIO Y, LODI A, PROUVOST A. Machine learning for combinatorial optimization: a methodological tour d'horizon[J]. European Journal of Operational Research, 2021, 290(2): 405 - 421.

[20] KHALIL E, DAI H, ZHANG Y Y, et al. Learning combinatorial optimization algorithms over graphs[C]// Proceedings of the 31st Conference on Neural Information Processing Systems, 2017.

[21] KOOL W, HOOF H V, WELLING M. Attention, learn to solve routing problems![C]//Proceedings of the 7th International Conference on Learning Representations, 2019.

[22] KWON Y D, CHOO J, KIM B, et al. POMO: policy optimization with multiple optima for reinforcement learning[C]//Proceedings of the 34th Conference on Neural Information Processing Systems, 2020.