第32卷 第2期
2022年2月
计算机技术与发展
COMPUTER TECHNOLOGY AND DEVELOPMENT
Vol. 32 No. 2
Feb. 2022

求解0-1背包问题的融合贪心策略的回溯算法

孙佳宁[1,2,3],马海龙[1,2,3],张立臣[1,2,3],李 鹏[1,2,3*]
(1. 现代教学技术教育部重点实验室,陕西 西安 710062;
2. 陕西省教学信息技术工程实验室,陕西 西安 710119;
3. 陕西师范大学 计算机科学学院,陕西 西安 710119)

摘 要:0-1背包问题作为经典的NP完全问题一直得到广泛的关注和研究。研究发现,经典回溯算法在解决0-1背包问题时的算法时间复杂度较高,尤其是在物品数量较多时,短时间内不能得到问题的解,导致算法的适用性较差。虽然经典贪心算法和现阶段涌现出的大量新型算法能够极大地缩减算法的运行时间,但普遍是以牺牲算法的准确性为代价的,不能保证可以找到问题的最优解。针对这些问题,提出一种融合贪心策略和剪枝策略的新型回溯算法。该算法将贪心算法得到的问题近似解用于剪枝策略的判断条件中,并在物品取舍时将当前的物品重量与背包的剩余容量进行比较,以避免重复计算,减少迭代次数,提高算法的执行效率。大量的仿真实验结果表明,在一定问题规模下,与经典回溯算法相比,所提出的新型回溯算法仍能够在短时间内准确找到问题的最优解,且具有更高的执行效率。

关键词:0-1背包问题;贪心算法;回溯算法;剪枝策略;递归算法
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2022)02-0190-06
doi:10.3969/j.issn.1673-629X.2022.02.031

Backtracking Algorithm of Fusion Greedy Strategy for Solving 0-1 Knapsack Problem

SUN Jia-ning[1,2,3],MA Hai-long[1,2,3],ZHANG Li-chen[1,2,3],LI Peng[1,2,3*]
(1. Key Laboratory of Modern Teaching Technology of Ministry of Education,Xi'an 710062,China;
2. Engineering Laboratory of Teaching Information Technology of Shaanxi Province,Xi'an 710119,China;
3. School of Computer Science,Shaanxi Normal University,Xi'an 710119,China)

Abstract:As a classic NP-complete problem,the 0-1 knapsack problem has been receiving extensive attentions and research. It is found that when using the classic backtracking algorithm to solve the 0-1 knapsack problem,the time complexity of the algorithm is relatively high,which leads to poor applicability when the number of the problem is large,which means that the solution of the problem cannot be obtained in a short time. Although the classic greedy algorithm and a large number of new algorithms being generated currently greatly reduce the running time,they generally sacrifice the accuracy of the algorithm and cannot guarantee that the optimal solution to the problem can be found. In order to solve these problems,a new backtracking algorithm combining greedy strategy and pruning strategy is proposed,in which an approximate solution of the problem obtained by the greedy algorithm is used as the judgment condition of pruning strategy,and comparing the current weight of item with the remaining capacity of the backpack when selecting items,so as to avoid repeated calculations,reduce the number of iterations,and improve the algorithm implementation efficiency. Extensive experiments are conducted to determine the optimal solution. The results show that,under a certain problem size,comparing with the classic backtracking algorithms,the proposed backtracking algorithm can accurately find the optimal solution to the problem in a shorter time,and thus achieves high execution efficiency.

Key words:knapsack problem;greedy algorithm;backtracking algorithm;pruning strategy;recursion algorithm

收稿日期:2021-03-29 **修回日期**:2021-07-29
基金项目:国家自然科学基金项目(61877037);教育部第二批新工科研究与实践项目(E-RGZN20201045);陕西师范大学基础教育课程研究中心项目(2019-JCJY009);陕西师范大学金课(算法设计与分析)建设项目(2019)
作者简介:孙佳宁(2000-),女,研究方向为计算机科学与技术;通信作者:李 鹏(1981-),男,博士,副教授,CCF会员(17213M),研究方向为物联网、媒体计算、教育信息科学与技术。

0　引　言

背包问题是指,在指定的背包容量下,如何选择总价值最大的物品装入背包。在背包问题中,若每个物品只能被选择一次,且装入时不可拆分,称为 0-1 背包问题。0-1 背包问题是组合优化问题中经典的 NP 完全问题[1],一直以来得到广泛的研究和应用[2-4],如货物装载、投资选择、密钥生成等[5]。

随着科技的发展和新型算法策略的不断涌现,0-1 背包问题的求解算法也由经典的蛮力法、贪心算法、动态规划法、回溯法、分支限界法等逐渐发展出众多新型智能算法,如萤火虫算法、量子狼群算法、烟花算法、混合蝙蝠算法等[6-15]。这些新型算法在一定程度上能够提高搜索能力和求解速度,但并不能保证找到问题的最优解,其算法的高效性是以牺牲算法的最优性为代价的。

在保证得到 0-1 背包问题最优解的前提下,该文以贪心算法和回溯算法为基准,以提高算法性能为目标,对其进行改进,提出了一种新型回溯算法。该算法利用了"回溯算法在搜索过程中判断、调试"的特点,并通过先运行得到的贪心算法的近似解用于经典回溯算法剪枝策略的判断条件,同时优化了该算法中针对物品选择的判断条件。

通过大量的仿真实验,如设置每个物品的重量和价值、背包的容量、背包可容纳的物品数量的最大上限等,设计对比实验,观察和分析了经典的回溯算法与该文提出的新型回溯算法找到问题最优解时的时间耗费,验证所提出算法的高效性和可靠性。

1　0-1 背包问题建模与经典算法求解

1.1　问题描述与形式化模型

经典的 0-1 背包问题指的是给定一些物品和一个背包,每个物品的重量和价值已知,背包的容量已知。选择一定的物品装入背包,在选择的过程中,要求每个物品只能被选择一次,且每次只有装入或不装入两种选项(即物品不可分割)。如何进行物品的选择,才能使放入背包的物品总重量在不超过背包容量的前提下,拥有最大的价值总和。

假设物品数量为 n ,背包容量为 C ,第 i 个物品的重量为 $w[i]$,价值为 $v[i]$,则 0-1 背包问题的形式化模型表示如下:

$$\max \sum_{i=1}^{n} v[i] \cdot x[i]$$

$$\text{s.t.} \sum_{i=1}^{n} w[i] \cdot x[i] \leqslant C$$

$$x[i] \in \{0,1\}, i \in \{1,2,\cdots,n\}$$

其中, $x[i]$ 表示物品 i 的选择状态, $x[i]=1$ 表示选中第 i 个物品放入背包, $x[i]=0$ 表示第 i 个物品没有放入背包。

1.2　0-1 背包问题的贪心算法

众所周知,贪心算法兼顾了问题求解的可行性和算法执行的高效性,是近似解决最优化问题较简单、较迅速的求解算法。在问题求解的过程中,贪心算法并不考虑问题的整体性,只考虑当前条件下的最优选择,通过局部最优解逐步构造出问题的解,因而往往只能得到问题的近似最优解。

常见的贪心策略有"物品重量最小优先"、"物品价值最大优先"和"物品单位价值最大优先"。在实际应用中,"物品单位价值最大优先"的贪心策略往往效果最好,因而该文采用该贪心策略,其主要步骤描述如下:

(1)计算每个物品的单位重量价值,并按照该单位价值以递减顺序对物品进行排序,排序后的结果存储到一维数组 index 中;

(2)根据排序后的结果,按照单位价值从大到小的顺序,依次将每个物品放入背包,直到背包内放不下新的物品为止;

(3)物品最终的选择状态存储到一维数组 result 中,背包内物品的最大价值存储到 maxvalue 中。

上述"物品单位价值最大优先"的贪心算法(greedy)的伪代码描述如下:

```
算法 1:贪心算法 greedy。
1. maxvalue←0
2. for i =1 to n do
3. index [i] ← i;
4. sort [i] ← v[i] ÷ w[i];
5. for i =1 to n do
6. for j =1 to n - i do
7. if sort[ j ]<sort[ j +1] then
8. sort[ j ]↔sort[ j +1];
9. index[ j ]↔index[ j +1];
10. for i =1 to n do
11. if w[index[ i ]]≤C then
12. x [index[ i ]]←1;
13. C ← C -w[index[ i ]];
14. else
15. break;
16. for i =1 to n do
17. maxvalue←maxvalue+ x[i] * v[i];
18. result[ i] ← x[i];
19. return maxvalue
```

算法 1 的时间复杂度主要由排序算法决定。由于经典冒泡排序算法的时间复杂度为 $O(n^2)$,贪心选择(代码 10 到 15 行)的时间复杂度为 $O(n)$,故算法 1 的时间复杂度为 $O(n^2)$,其中 n 为物品的数量。若将

排序算法改为快速排序算法,可以将该算法的时间复杂度降低为 $O(n\log n)$ 。

1.3 0-1 背包问题的经典回溯算法

与贪心算法相比,经典回溯算法更看重问题的整体性分析。经典回溯算法是使用深度优先遍历求解 0-1 背包问题的经典算法[16]。在问题求解的过程中,回溯算法将物品的选择状态构造成一个解空间树,通过遍历解空间树中的每个节点来寻找问题的可行解和最优解。具体地说,经典的回溯算法从解空间树的根节点出发,依次判断解空间树的每一个节点,并选择当前状态下满足问题约束条件的节点,当遍历到叶子节点时,算法对当前各个节点的选择状态进行记录和判断,当约束条件不满足时,回退一步至上一状态或回退多步,直至遍历过解空间树的每一个节点并回溯到根。在搜索的过程中,回溯算法会不断判断是否存在更优的解,并记录和更新目前已找到的最优解。

经典回溯算法解决 0-1 背包问题的主要步骤如下:

(1)构造问题的解空间树;

(2)确定问题的约束条件。一个是放入背包的物品总重量不超过背包的容量,另一个是放入背包的物品是否构成问题的最优解;

(3)从根节点开始,深度优先遍历解空间树,在遍历的过程中根据问题的约束条件选择性地进行回溯和继续遍历。

在使用经典回溯算法解决 0-1 背包问题时,最坏情况下需要遍历整个解空间树,此时算法的时间复杂度为 $O(2^n)$ 。但一般情况下,回溯算法总会在遍历到解空间树的最后一个节点前找到问题的解,在实际的遍历过程中,算法的运行时间取决于遍历时生成的节点数目,即在找到问题的最优解时,算法所遍历到的节点数目。

1.3.1 递归回溯算法

存在两种实现经典回溯的算法,一个是递归算法,一个是非递归算法。递归回溯算法,是指把一个大型的复杂问题分解为一个与原问题相似的、规模较小的问题,通过递归求解小问题并将子问题的解合并,从而得到原问题的解。

算法 2 给出了递归回溯算法(knap1)的伪代码描述,该算法从位于解空间树根节点的第 1 个物品开始遍历,依次考虑当前选中物品不放入背包和放入背包时,所有物品的选择状态。当遍历到叶子节点时,进行最优解的约束条件检验并判断是否进行回溯,算法的主要步骤如下:

(1)引入 cw 和 cv 分别表示当前状态下背包内物品的总重量和总价值;

(2)对于每一个物品,首先选择“不装入背包”并进行递归,而在进行“装入背包”的选择前,则需要判断所有选中物品的总重量是否满足背包的容量;

(3)当遍历到最后一个物品时,计算此时背包内物品的总价值,判断是否为问题的最优解。

算法 2:递归回溯算法 knap1。

```
1. Initialize maxvalue, cw, cv as 0
2. knap1(1)
3. output maxvalue
   int knap1(int i)
1. if i = n +1 then
2. if cv > maxvalue then
3. maxvalue ← cv;
4. result← x;
5. else
6. x[i] ←0;
7. knap1( i + 1);
8. if cw+ w[i] ≤C then
9. x[i] ←1;
10. cv←cv+ v[i], cw←cw+ w[i];
11. knap1(i +1);
12. x[i] ←0;
13. cv←cv- v[i], cw←cw- w[i];
14. return maxvalue
```

1.3.2 非递归回溯算法

与递归回溯算法不同,非递归回溯算法利用物品的状态值判断是否需要回溯,而不需要借助操作系统提供的递归机制,因而一般具有较高的时空效率。

算法 3 给出了非递归回溯算法(knap2)的伪代码描述,在该算法中,选中物品在进行“装入”或“不装入”背包的选择前,其状态值首先会被赋值为-1(代码第 2 行);当物品的选择状态确定后,其状态值则为 0 或 1(代码第 4、15 行);一旦该物品的状态值增加至 2 时,则需要进行回溯(代码第 17、18 行)。

算法 3:非递归回溯算法 knap2。

```
1. Initialize maxvalue as 0, i as 1
2. x[i] ←-1
3. while i ≥1 do
4. x[i]++;
5. if x[i] ≤1 then
6. if i = n then
7. for j =1 to n do
8. cw←cw+ x[j] * w[j];
9. cv←cv+ x[j] * v[j];
10. if cw≤ C and cv>maxvalue then
11. maxvalue←cv;
12. result← x;
13. cw←0, cv←0;
14. else
```

```
15. i ++;
16. x[i] ←-1;
17. else
18. i --;
19. end
20. return maxvalue
```

2　融合贪心策略和剪枝策略的回溯算法

2.1　问题分析

使用贪心算法求解 0-1 背包问题时,算法从问题的某一初始解出发,选择当前条件下的最优解,并试图构造或逼近问题的整体最优解。贪心算法每一步的决策仅考虑当前的局部信息,其解空间不可回溯再现,具有较强的随机性和不可预知性。这种基于经验或直觉的判断,并不一定能够保证找到问题真正的最优解,在绝大多数情况下,贪心算法得到的解只是问题的近似最优解。但是,由于贪心算法采用局部最优决策,因而往往具有较高的效率。

回溯算法的思想和枚举法类似,即通过尝试问题所有可能的解来寻找问题的最优解,这种算法虽然确保了解的正确性,但是以牺牲算法运行时间为代价。当问题规模较大时,回溯算法的运行时间呈指数式增长,在短时间内可能无法得到问题的最优解。

针对上述特征,该文将贪心算法的高效性和回溯算法的最优性相结合,提出了一种融合贪心策略和剪枝策略的新型回溯算法,简称新型算法。考虑到贪心算法得到的解虽然可能不是问题的最优解,但至少一定是问题的近似解,故将其作为初始解应用于回溯算法,从而可以尽快实现剪枝操作,进而提高回溯算法效率。

因此,新型算法将贪心算法的解作为回溯算法中剪枝策略的判断条件,同时优化了遍历过程中的约束条件,从而在确保得到问题最优解的同时,可以进一步提高算法效率。

2.2　新型算法

算法 4 给出了所提出的新型算法的伪代码描述。该算法使用递归方法求解 0-1 背包问题,与经典的回溯算法不同,首先,算法 4 增加了基于价值的剪枝策略,若背包内物品的总价值与未遍历到的物品的总价值之和小于算法 1 得到的解,则进行剪枝并回溯。其次,在遍历的过程中,算法 4 优化了物品选择的约束条件,将经典回溯算法的约束条件“放入背包的物品总重量不超过背包的容量”修改为“当前选中的物品重量满足背包的剩余容量”。

所提出新型算法的主要步骤如下:

(1)引入 pw 和 pv 分别表示前 $i-1$ 个物品中,已经放入背包的物品的总重量和总价值,rw 表示背包的剩余容量,rv 表示未遍历到的物品的总价值(包含当前物品);

(2)算法 1 得到的解存储到 maxvalue_greedy 中;

(3)定义递归函数 GB(i, rw, pw, rv, pv),从初始状态 GB(1, C, 0, totalvalue, 0)开始递归;

(4)首先判断选中物品“不装入背包”时是否需要剪枝;

(5)若选中物品“装入背包”,首先判断物品重量是否满足背包的剩余容量,若“能装入”,则继续进行步骤 4 的剪枝判断,最终确定物品的选择状态。

算法 4:新型算法。

```
1. Initialize maxvalue, pw, pv as 0, rw as C, rv as totalvalue, i as 1
2. maxvalue_greedy←greedy( )
3. GB(1, C, 0, totalvalue, 0)
4. output maxvalue
   int GB(int i, int rw, int pw, int rv, int pv)
1. if i = n +1 then
2. for i to n do
3. value←cv+ v[i] * x[i];
4. if value>maxvalue then
5. maxvalue←value;
6. result← x;
7. else
8. if pv+rv- v[i] >maxvalue_greedy then
9. GB( i +1, rw, pw, rv- v[i], pv);
10. if w[i] ≤rw then
11. if pv+rv>maxvalue_greedy then
12. x[i] ←1;
13. GB( i +1, rw- w[i], pw+ w[i], rv- v[i], pv+ v[i]);
14. x[i] ←0;
15. return maxvalue
```

在新型算法中,递归函数 GB(第 3 行)的含义是:若当前考虑的物品“不装入背包”,计算此时背包内物品的总价值与未遍历到的物品的总价值之和,若大于 maxvalue_greedy,则表明,在当前物品的选择状态下,存在将某些物品装入背包后使得总价值提高的可能,因此进行递归;否则,表明在当前物品的选择状态下,任何后续的装入选择都将不可能超过现在背包内物品的总价值,此时则进行剪枝并回溯。若当前选中物品想要“装入背包”,首先判断该物品的重量是否满足背包的剩余容量,然后继续判断是否“有必要装入”。当遍历到最后一个物品时,计算此时背包内物品的总价值并判断是否为问题的最优解。

新型算法的核心思想是:对于每一个物品,只有该物品“能装入背包”且“有必要装入”,才会将其放入背包中。

3 仿真实验与分析

3.1 实验环境

该文进行了大量的仿真实验，具体实验环境为：CPU：i7-8550U，内存：DDR2 16 GB；硬盘：1 TB；缓存，8 MB；PF 使用率：36% ~45%；编程语言：Java，软件开发工具：Eclipse。分别实现了递归回溯算法（算法 2）、非递归回溯算法（算法 3）和新型算法（算法 4），并在不同问题规模下进行仿真实验与分析。

实验参数设置如下：根据物品数量 n，算法按照均匀分布随机地产生 1 ~20 之间的整数作为每个物品的重量和价值，背包容量为物品总重量与一个预先定义的实验系数 f 之积，其中，f 为一个 0 到 1 之间的小数，f 越小，表示背包的容量越小。

3.2 实验结果分析

为验证不同算法在不同问题规模下的运行时间，该文取实验系数 f 为 0.3、0.6、0.8，在不同问题规模 n 下，使用递归回溯算法（算法 2）、非递归回溯算法（算法 3）和新型算法（算法 4）得到问题最优解时的时间耗费进行实验。实验结果依次见表 1 ~ 表 3。

表 1 三种算法在实验系数 f =0.3 的时间耗费

物品数量 n	时间耗费/ms		
f =0.3	递归回溯算法	非递归回溯算法	新型算法
10	0.6	1	0.6
20	1.6	119.2	1.4
25	19.4	4 110	4.6
30	382.3	168 279	59.2

表 2 三种算法在实验系数 f =0.6 的时间耗费

物品数量 n	时间耗费/ms		
f =0.6	递归回溯算法	非递归回溯算法	新型算法
10	0.4	1.2	0.6
20	10.6	118.2	1.8
25	296.2	4 158	5
30	5 591	170 423	17.2

表 3 三种算法在实验系数 f =0.8 的时间耗费

物品数量 n	时间耗费/ms		
f =0.8	递归回溯算法	非递归回溯算法	新型算法
10	0.6	1.3	0.5
20	14.5	131	1.9
25	352	4 275	2.3
30	6 078	85 064	3.9

通过上述比较，可以得出以下结论：

（1）在不同实验系数下，随着问题规模的增加，所有算法的时间耗费都呈现上升趋势，其中经典的递归回溯算法和非递归回溯算法的上升趋势更为显著。结合算法的时间复杂度可知，经典回溯算法的时间耗费随着问题规模的增加呈指数级增长。此外发现，非递归回溯算法较递归回溯算法具有较高的运行时间。

（2）在同一实验系数、同一问题规模下，与经典回溯算法相比，该文所提出的新型算法在保证得到问题的最优解的同时，具有较小的时间耗费，特别是在问题规模较大时，效果更为明显。

在不同实验系数下，对递归回溯算法（算法 2）和非递归回溯算法（算法 3）的时间耗费进行了对比，从而验证实验系数是否对经典回溯算法的执行时间产生影响。实验结果见表 4。

表 4 经典回溯算法在问题规模 n =20 时的时间耗费

f	递归回溯算法	非递归回溯算法
0.3	1.6	119.2
0.4	5	121.6
0.5	8	117.8
0.6	10.6	118.2
0.7	14.2	118.5
0.8	14.8	118.8
0.9	14.5	121.4

根据表 4 可知，当实验系数从 0.3 逐渐增加至 0.9 时，递归回溯算法的时间耗费线性增长至某一数值后，在该数值处呈现小幅度波动；非递归回溯算法的时间耗费则始终在某数值处小幅度波动。

从表中可知，当实验系数增大至 0.7 时，两种回溯算法的时间耗费都基本保持稳定，故取实验系数 f = 0.8，单独对新型算法进行实验，分析其算法运行时间与问题规模间的关系。实验结果如图 1 所示。

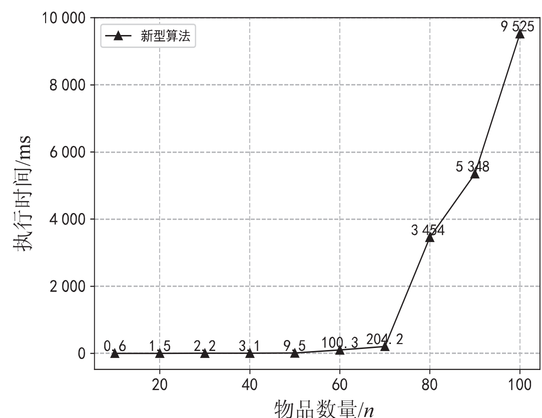

图 1 新型算法的时间耗费（f =0.8）

从图 1 可以看出，在实验系数为 0.8 时，只有当问题规模近似增加至 70 时，新型算法的时间耗费才会有较为明显的递增，效率远远高于经典回溯算法。

表 5 列出了在不同的实验系数和不同的问题规模下，新型算法得到问题最优解的时间耗费情况。从表 5 中可知，新型算法普遍随实验系数的增加而减少，其

原因在于,随着实验系数的增加,背包的容量不断增大,背包可以容纳的物品数量相应增加,从而减少了回溯的概率,进而提高了算法效率。

表 5　新型算法在不同实验系数、不同问题规模下的时间耗费

f	10	30	50	75	100
0.3	0.9	63.2	434 349	-	-
0.4	1.1	41.6	234 892	-	-
0.5	0.8	35.2	119 993	-	-
0.6	0.8	14.5	97 435	-	-
0.7	0.6	7.2	1 378	-	-
0.8	0.6	4.7	86.1	1 760	226 680
0.9	0.6	2.2	9.5	204	9 525

4　结束语

0-1 背包问题是经典的 NP 完全问题,而经典的回溯算法采用枚举的思想,通过深度优先搜索解空间树寻找问题的最优解。在问题规模较小时,回溯算法同其他求解 0-1 背包问题的经典算法相比,具有更小的时间耗费和空间耗费。但随着问题规模的增加,回溯算法在时间和空间上都有极为明显的增加,这种增加导致算法在较短时间内无法得到问题的最优解。

该文将贪心算法的高效性和经典回溯算法的最优性相结合,提出的融合贪心策略与剪枝策略的新型算法,将经典贪心算法得到的问题近似解用于剪枝策略的判断条件中,减少了经典回溯算法遍历过程中的无效搜索;同时,在进行物品选择的判断时,使用背包剩余容量作为约束条件,大大提高了算法的求解效率。大量仿真实验结果表明,该新型算法在保证得到问题最优解的同时,有效提高了算法的运行效率。

参考文献:

[1] MERKLE R,HELLMAN M. Hiding information and signatures in trapdoor knapsacks[J]. IEEE Transactions on Information Theory,1978,24(5):525-530.

[2] 杨新木,杨　静,殷志祥,等. DNA 折纸术在 0-1 背包问题中的应用[J]. 计算机应用研究,2021,38(3):777-781.

[3] BORGULYA I. An EDA for the 2D knapsack problem with guillotine constraint[J]. Central European Journal of Operations Research,2019,27:329-356.

[4] OZSOYDAN F B,BAYKASOGLU A. A swarm intelligence-based algorithm for the set-union knapsack problem[J]. Future Generation Computer Systems,2019,93:560-569.

[5] 朱阅岸. 解 0-1 背包问题的算法比较和改进[D]. 广州:暨南大学,2011.

[6] 任静敏,潘大志. 带权重的贪心萤火虫算法求解 0-1 背包问题[J]. 计算机与现代化,2019(5):86-91.

[7] 严雅榕,项华春,聂　飞,等. 求解 0-1 背包问题的量子狼群算法[J]. 微电子学与计算机,2018,35(7):1-5.

[8] 徐小平,庞润娟,王　峰,等. 求解 0-1 背包问题的烟花算法[J]. 计算机系统应用,2019,28(2):164-170.

[9] XUE Junjie,XIAO Jiyang,ZHU Jie. Binary fireworks algorithm for 0-1 knapsack problem[C]//2019 international conference on artificial intelligence and advanced manufacturing (AIAM). Dublin,Ireland:AIAM,2019.

[10] 万晓琼,张惠珍. 求解 0-1 背包问题的混合蝙蝠算法[J]. 计算机应用研究,2019,36(9):2579-2583.

[11] 王建辉,郑光勇,徐雨明. 混合人工化学反应优化算法求解 0-1 背包问题[J]. 计算机技术与发展,2020,30(7):71-75.

[12] ALZAQEBAH A,ABU-SHAREHA A A. Ant colony system algorithm with dynamic pheromone updating for 0/1 knapsack problem[J]. International Journal of Intelligent Systems and Applications,2019,11(2):9-17.

[13] ABDEL-BASSET M,EL-SHAHAT D,SANGAIAH A K. A modified nature inspired meta-heuristic whale optimization algorithm for solving 0-1 knapsack problem[J]. International Journal of Machine Learning and Cybernetics,2019(3):495-514.

[14] LAABADI S,NAIMI M,EL AMRI H,et al. An improved sexual genetic algorithm for solving 0/1 multidimensional knapsack problem[J]. Engineering Computations,2019,36(7):2260-2292.

[15] 陈　桢,钟一文,林　娟. 求解 0-1 背包问题的混合贪婪遗传算法[J]. 计算机应用,2021,41(1):87-94.

[16] 王晓东. 算法设计与分析[M]. 第 3 版. 北京:清华大学出版社,2014.