考虑时间因素的0-1背包调度问题

王正理[1,3] 谢 添[2] 何 琨[1,3] 金 燕[1,3]
(华中科技大学计算机科学与技术学院 武汉 430074)[1]
(南加州大学维特比工程学院 洛杉矶 90089)[2] (深圳华中科技大学研究院 广东 深圳 518057)[3]

摘 要 文中提出考虑时间因素的0-1背包调度问题这一具有NP难度的组合优化问题。给定 n 个物体(每个物体 i 的重量为 w_i,连续加工时间为 t_i),以及一个容量为 S 的背包,要求给出一个调度方案(物品的放入顺序和放入时间),使得任意时刻放入背包的物品总重量不超过背包容量,每个物体需放入背包连续加工时长 t_i 后才能取出,该问题是求使所有物体均加工完毕的时间尽可能短的调度方案。提出了3种求解算法:迭代动态规划算法、基于分枝限界的完备算法和遗传进化算法。迭代动态规划算法使用动态规划策略放置尽可能多的未加工物体到背包中,然后每次迭代取出加工完成的物品后再使用动态规划放入尽可能多的剩余未加工物品,直至所有物品被加工完成。基于分枝限界的完备算法通过定义上下界及剪枝操作,有效地降低了算法的计算复杂度。遗传进化算法将一个物品装填序列定义为个体,并定义了相应的适应度、选择、交叉与变异操作。在所设计的3组共计36个算例上的实验结果表明,迭代动态规划算法可以很快求出高质量的解,基于分枝限界的完备算法对小规模算例有很好的效果,遗传算法在处理几百个物体的算例时能在1500 s内得到比动态规划算法更好的结果。

关键词 背包调度,动态规划,分枝限界,遗传算法
中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.04.007

0-1 Knapsack Variant with Time Scheduling

WANG Zheng-li[1,3] XIE Tian[2] HE Kun[1,3] JIN Yan[1,3]
(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)[1]
(Viterbi School of Engineering, University of Southern California, Los Angeles 90089, USA)[2]
(Shenzhen Research Institute of Huazhong University of Science and Technology, Shenzhen, Guangdong 518057, China)[3]

Abstract This paper proposed an NP-hard 0-1 knapsack variant problem considering the space and time issues. Given n items with each item i having weight w_i and continuous storage time length t_i, and a knapsack with capacity S, a scheduling is asked to provide for the packing time of each item(the removing time can be deduced accordingly). At any time instant, the total weights of the packed items should not exceed the capacity of the knapsack. This paper proposed three algorithms to address this problem: a quick dynamic programming(DP) algorithm, a branch and bound(BnB) based exact algorithm and a genetic algorithm. The dynamic programming(DP) algorithm first regards all items as raw items, and uses DP to pack as much raw items as possible into the knapsack. Whenever there is an item that becomes mature, i.e., it has been stored enough time in the knapsack, the mature item is removed from the knapsack, and for the remaining Knapsack capacity, DP is used again to pack as much raw items as possible. The above process continues until all items are mature and removed from the knapsack, completing the DP scheduling. The BnB based exact algorithm defines the lower bound and upper bound, and cuts the unnecessary branches to shorten the running time. The genetic algorithm defines each individual as a packing order, and defines the corresponding fitness value, selection, mutation and crossover operators. Experiments on three sets with a total of 36 designed benchmark instances show that DP can quickly produce high quality schedules, BnB based exact algorithm works well for small instances, the genetic algorithm can deal with hundreds of items within 1500 seconds and it outputs schedules with considerably shorter makespan when compared with DP.

Keywords Knapsack scheduling, Dynamic programming, Branch and bound, Genetic algorithm

---

到稿日期:2017-07-02 返修日期:2017-08-12 本文受国家自然科学基金项目(61472147,61602196,61173180),深圳市科技计划项目(JCYJ20170307154749425)资助。

王正理(1994-),男,硕士,主要研究方向为NP难问题的算法设计;**谢 添**(1994-),男,硕士,主要研究方向为算法设计和分析;**何 琨**(1972-),女,教授,主要研究方向为算法设计和分析、数据挖掘和机器学习,E-mail:brooklet60@hust.edu.cn(通信作者);**金 燕**(1986-),女,讲师,主要研究方向为NP难问题和大规模组合优化问题的启发式算法求解。

在经典 NP 难度问题 0-1 背包问题的基础上，提出了考虑时间因素的 0-1 背包调度问题(0-1 Knapsack Variant with Time Scheduling，KVTS)。该问题可被描述为：已知一个承重为 S 的背包和 n 个重量为 $w_1, w_2, \cdots, w_n$ 的物品($\max_i w_i \leqslant S \ll \sum_1^n w_i$)，每个物品需放入背包连续加工时长 t_i，且任意时刻背包内物体的总重量不能超过背包总承重。要求给出一种调度，使得从第一个物品放入到最后一个物品取出的时间跨度即 *makespan* 尽可能小。

KVTS 是在 0-1 背包的基础上增加了时间因素的高复杂度的 NP 难度问题，在现实生活中有着广泛的应用。如果把背包看成一台机器，将各个物品看成待连续加工的工件，则求加工完所有物品的最优调度问题可被转化为一个单机有容量限制的作业调度问题。若将时间简单视为空间，则 KVTS 可退化为二维矩形条带 packing 问题[1]。对于将 0-1 背包问题拓展到二维、三维，加上时间因素而构成的三维、四维装箱调度问题的研究，黄文奇和何琨[2-5]提出了可行算法，并给出了其可计算性证明。

背包问题(Knapsack Problem，KP)自 1978 年被 Merkel 和 Hellman 提出以来[6]，一直受到广泛研究。解决背包问题的算法主要分为两类：精确算法[7-9](如分枝限界法、回溯法等)和启发式算法(如近似算法[10-11]、遗传算法[12-13]、蚁群算法[14])。精确算法一般针对小规模算例，能在短时间内得到问题的精确解，而启发式算法则对大规模算例的计算有显著效果。对于二维矩形条带 Packing 问题，较有代表性的算法可分为随机型算法和确定型算法两大类。随机型算法主要包括局部搜索算法[15]、遗传算法[16]、模拟退火算法[17]等；确定型算法主要包括分枝限界法[18]、贪心算法[19]、BL(bottom-left)[20]算法、BLF(bottom-left-fill)[21]算法、BLD(bottom-left-decreasing)[22]算法等。还有学者将上述两类方法结合，以进一步提高求解的质量[23]。

本文提出了 3 种算法来求解 KVTS 问题：迭代动态规划算法、基于分枝限界的完备算法以及遗传进化算法。对这 3 种算法在所设计的不同规模算例上进行实验，以分析评估其求解的性能差异。

1 迭代动态规划算法

1.1 算法的基本思路

由于物品的总重量远大于背包的容量，在每次装填时，使用动态规划来尽可能地“装满”背包。这里，“装满”可以有多种定义，如只考虑物品重量，也可以考虑物品的重量与时间的乘积等。当剩余时间最短的装入物品被取出后，再对背包剩余容量进行下一次装填。

整个算法的核心是建立准确的动态规划递归方程。每次装填时，假设 $c[i][j]$ 表示未装入的前 i 个物品在背包容量不超过 j 的情况下所能达到的最优值，S 为背包的初始总容量，则：

1)考虑每次装填使物品总重量最大的动态规划递归方程，如式(1)所示。

2)考虑每次装填使物品最大的动态规划递归方程，如式(2)所示。

$$c[0][j]=0, 0\leqslant j\leqslant S$$
$$c[i][j]=\begin{cases}\max(c[i-1][j], c[i-1][j-w_i]+w_i), & w_i\leqslant j\leqslant S\\ c[i-1][j], & 0\leqslant j<w_i\end{cases} \quad (1)$$

$$c[0][j]=0, 0\leqslant j\leqslant S$$
$$c[i][j]=\begin{cases}\max(c[i-1][j], c[i-1][j-w_i]+w_i*t_i), & w_i\leqslant j\leqslant S\\ c[i-1][j], & 0\leqslant j<w_i\end{cases} \quad (2)$$

当各物品的装填时间相等时，该问题便退化为经典的 0-1 背包问题，动态规划算法可以得到理论最优解，且上述两个策略得到的解相同。当各物品的装填时间不等时，通常第二个策略的效果更好。

1.2 算法流程和时间复杂度分析

初始时刻 $T\leftarrow 0$，初始输入列表为全部物品的集合 X。然后计算当前未装入物品的总重量，记为 *totalweight*。若 $S\geqslant$ *totalweight*，则当前剩余物品可以一次装入，将所有物品标记为已装入，取所有物品中加工时间的最大值 t_m，$T\leftarrow T+t_m$，结束。否则，不能一次全部装入，利用动态规划算法使得装入的物品的 $\sum w_i * t_i$ 最大，其中 w_i 和 t_i 分别为物品 i 的重量和加工时间。此时，背包当前的可承重 S 须减去本轮所有装入物品的重量。取出背包内所有剩余时间 t_{ri} 最短的物品，记其所需时间为 $t_{\min}$，$T\leftarrow T+t_{\min}$，将仍在背包内的物品的剩余时间都减去 $t_{\min}$，并将 S 加上此刻所有取出物品的总重量。将未装入物品更新为新的输入列表，再进行下一轮的装填过程。如此循环迭代，直至装填结束。算法的伪代码如算法 1 所示。

算法 1 迭代动态规划算法(IDP)

```
输入：背包承重 S，物品数量 n，物品的集合 X
输出：所需的时间 T
1. T←0；
2. initialize(X)；
3. totalweight←∑_{i=1}^{n} w_i；
4. while S<totalweight do
5.    dp(X,S,n)；/ * 动态规划 * /
6.    update(S)；
7.    t_min← min_i t_ri；/ * t_ri为当前背包内物体的剩余加工时间 * /
8.    T←T+t_min；
9.    t_r←t_r−t_min；/ * 更新当前背包内物体的剩余加工时间 * /
10.   update(S)；
11.   update(X)；
12.   totalweight←∑w_i；/ * 剩余未加工物体总重 * /
13. end while
14. t_m←max_i t_i； / * 找到剩余未加工物体的最长加工时间 * /
15. T ← T+t_m；
16. return T；
17. end algorithm
```

对于一轮动态规划装填，需要最多计算 nS 个 dp 状态，计算每个状态的时间复杂度是 O(1)；计算状态后可以得到需要装填的物品，装填操作的时间复杂度是 O(n)。一次动态规划状态的时间复杂度为 O(nS)+O(n)=O(nS)。共有 n 个物品，最多需要 n 次动态规划装填，因此算法总的时间复杂度为 nO(nS)=O(n^2S)。

1.3　与 2D Strip Packing 问题的比较

如果将每个物品的连续加工时间看成物体的高，将背包的容量看成容器的底，则求解整个考虑时间因素的 0-1 背包问题的最短任务时间就变成了求解这个容器能容纳所有物品的最小高度，这时该问题就退化为了 2D Strip Packing 问题。

由于考虑时间的背包问题只需要背包的剩余容量能够容纳要放入的物品就可以将物品放入，不需要考虑物品在背包中放置的位置，即可以认为任意时刻物品在背包中的位置都是可以挪动的。而在 2D Strip Packing 问题中，物品放入后就不能挪动，导致剩余空间无法整合到一起，在某一较低高度下考虑时间的背包问题中可以放入的物品无法得到放置。因此，考虑时间因素的 0-1 背包调度问题的实际结果会优于将其看成 2D Strip Packing 问题所得到的结果。

如图 1 所示，要放入第 4 号物品，当把问题看成 2D Strip Packing 问题时要把它放在 2 号物品上面，因为 1 号、3 号物品上面都没有足够的放置空间，即等效于要把 1，2，3 这 3 个物品都拿出后才能进行 4 号物品的放置。而实际上，对于本问题，在 1 号和 3 号物品都被拿出后，它们所占用的空间会释放并整合，足够放下 4 号物品，则 4 号物品的放置位置实际上在原来 1 号物品的顶部，总装载高度会比 2D Strip Packing 低。

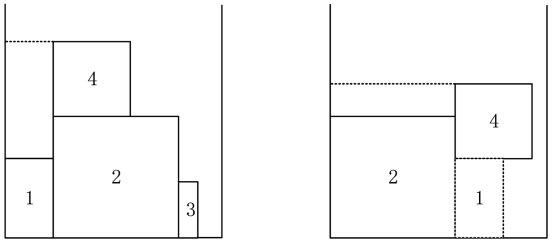

图 1　与 2D Strip Packing 问题的对比

Fig. 1　Comparison with the 2D Strip Packing problem

2　基于分枝限界的完备算法

求解 KVTS 的完备算法的基本思想为：对于一个给定的物品装填顺序，可以确定出装填这些物品所需的总时间。对于 n 个物品，由排列组合可知物品的装填序列共有 $n!$种。这些装填序列可以对应一个解空间树，遍历整个解空间树可得到问题的理论最优解。由于遍历整个解空间树的时间复杂度为指数级，本文提出了基于分枝限界的完备算法(Branch and Bound based exact algorithm，BnB)，根据对所有节点上下界的估计，在一些节点的子节点被遍历之前就进行剪枝操作。

通过构造解空间树能够更清楚地了解分枝限界的执行过程。该问题的解空间树如图 2 所示。从解空间树第 2 层到第 $n+1$ 层的序列就是一个完整的装填顺序序列，整棵树有 $n!$个序列。

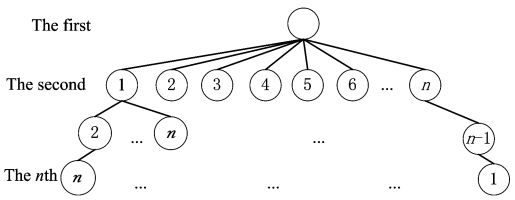

图 2　考虑时间因素的 0-1 背包问题的解空间树

Fig. 2　Solution space tree of 0-1 knapsack problem considering placing time

对于剪枝策略，通过估算当前节点所有分枝对应的序列子集的调度时间长度的上界 T_{ub} 和下界 T_{lb} 进行剪枝。在根节点，对整个物品集合使用迭代动态规划算法，得到的调度结果作为根节点的 T_{ub}，并把该时间上界作为已找到的历史最优解 T_{best}。根节点的 $T_{lb}=\max\{T_{max},T_{avg}\}$，$T_{max}$ 为所有物品中所需的最长加工时间；T_{avg} 为将所有物品的时间简单视为高来计算总面积，再除以容器容量的值。

若当前节点对应分枝的下界 T_{lb} 不超过历史最优解，则认为这个节点是合法的，只有合法节点才进一步展开搜索；否则进行剪枝，回溯到其父节点。当某个节点的 $T_{ub}=T_{lb}$ 时，将历史最优解 T_{best} 更新为 T_{lb}，并对该节点剪枝，回溯到其父节点。

对于某个非根节点 i，其 $T_{ub}(i)$ 和 $T_{lb}(i)$ 的计算如下：设节点 i 在第 k 层，即其对应一个数量为 $k-1$ 的装入物品序列，该物品序列当前的背包剩余容量为 S'，未加工物品的数量为 n'，当前的装填时间为 T_i。将对剩余未装入物品执行动态规划算法得到的时间加上 T_i 作为 $T_{ub}(i)$。下界的计算方法为：设剩余未装入物品中连续加工时间最长的物品为 m_1，其加工时间为 t_{m_1}，若 $n'>n/2$，剩余物品数量较多时，$T_{lb}(i)=T_i+\max\{t_{m_1},T_{avg}'\}$，$T_{avg}'=\sum w_i t_i/s$(剩余未加工物体的重量与时间的乘积的和除以背包总容量)。若 $n'\leqslant n/2$，背包中的剩余加工时间最长的物品为 m_2，其剩余加工时间为 t_{rm_2}，如果 m_1 的重量 $w_{m_1}\leqslant S'$，则 $T_{lb}(i)=T_i+\max(t_{m_1},t_{m_2})$；如果 $w_{m_1}>S'$，则将背包内物品按剩余加工时间从短到长依次拿出，直到背包的剩余容量刚好能容纳 m_1，此刻的装填时间为 T_i'，背包内物体的剩余最长加工时间为 t_{rm_2}'，$T_{lb}(i)=T_i'+\max(t_{m_1},t_{rm_2}')$。

假定当前节点 i 的分枝都被计算完毕，则需回溯至当前节点 i 的父节点，然后继续考虑父节点的下一个子节点，即 i 的兄弟节点。重复上述操作，直到所有的节点都计算完毕或者被剪枝，此时 T_{best} 即为理论最优解。

整个过程的伪代码如算法 2 所示。

算法 2　基于分枝限界的完备算法(BnB)

```
输入：背包承重 S，物品数量 n，物品的集合 X，当前装填深度 dep
输出：问题的理论最优解 Tbest
1. Tbest← IDP(S,n,X);
2. T[0] ← 0; /* T 为每个节点的当前装填时间 */
3. initial mark; /* mark[i]为遍历标记，初始都为 false */
4.   for i ← 1 to n do
5.   begin
6.     if mark[i]=false then
7.       vis[dep] ← I;/* vis 用来存储物品装填序列 */
8.       mark[i] ← true;
9.       t← T[dep−1];
10.      update(t);/* 装入 vis[dep]对应的物品不拿出，更新 T */
11.      T[dep] ← t;
12.      if dep>n/2 then
13.        if w_m1 ≤S' then/* m1 为剩余未装入物品中连续加工时
           间最长的物品 */
14.        T_lb←t+max(t_m1,t_rm2);/* m2 为背包中的剩余加工时
           间最长的物品/
15.      else
16.        T_lb←T_i'+max(t_m1,t_m2');
17.      endif
```

```
18.     else
19.       T_lb(i)←t+max{t_mi, Σw_i t_i/S};/* t_mi 为剩余物品的最长
          加工时间,Σw_i t_i 为所有剩余未加工物体的重量与时间的
          乘积的和 */
20.     endif
21.     if T_lb>=T_best then
22.       return
23.     endif
24.     T_ub←IDP(S',n-dep,X');/* S'为当前背包剩余容量,X'为
          剩余未装填物品 */
25.     if T_lb=T_ub then
26.       T_best← T_lb;
27.       return
28.     endif
29.     BnB(dep+1);
30.     update(vis);/*将物品装填还原至 vis[dep]对应的物品未被
          装填时的情况 */
31.     mark[i] ← false;
32.   endif
33. end
34. end algorithm
```

BnB 算法通过深度优先搜索对序列进行遍历,通过分枝限界进行剪枝,提高了效率。分枝限界的关键是估算当前节点的装填时间的上界和下界。当一个节点的装填时间的下界超过历史最优时,进行剪枝并回溯;若其装填时间的下界等于上界,则找到当前节点的最优装填时间,进行剪枝并回溯,更新历史最优解;若其时间下界不等于上界时,进行深度优先搜索,访问子节点。利用基于分枝限界的完备算法可以得到问题的理论最优解,但其只能在可接受时间内计算出较小算例的结果。

3 遗传进化算法

遗传进化算法(Genetic Evolution Algorithm)[24]是一类由生物界的进化规律(适者生存、优胜劣汰的遗传机制)演化而来的启发式搜索方法,被广泛用于组合优化、机器学习、信号处理等领域。遗传算法的关键在于如何将一个问题的解转化为遗传算法所能处理的个体,其搜索能力的核心在于对选择、交叉和变异 3 种操作的设计。针对 KVTS 问题,将遗传算法中个体的编码用物品的装填序列来表示,从而设计出解决 KVTS 问题的一种有效的遗传算法。

3.1 序列评估

为了评估遗传算法中个体的优劣,需要根据当前的物品装填序列来计算所需的装填时间。这一过程是遗传算法中的关键步骤,如算法 3 所示。

算法 3 序列评估(SE)

```
输入:背包承重 S,物品数量 n,物品的顺序 X
输出:装填时间 T
1. T←0;
2. for i ← 1 to n do
3.   begin
4.     if S>=w_i then
5.       S←S-w_i;
6.     else while S<w_i do
7.       t_min←min_i t_ri;/* t_ri为当前背包内物品的剩余加工时间 */
8.       T ←T+time;
9.       t_r← t_r-t_min;/* 更新当前背包内物体的剩余加工时间 */
10.      update(S);
11.      end while
12.    endif
13.   S ← S-w_i;
14. end
15. if i= n then /*装入到第 n 个物品 */
16.  t_m←max_i t_i;/*找到剩余未加工物体的最长加工时间 */
17.  T ←T+t_m;
18. endif
19. initialize(X);
20. initialize(S);
21. return T;
22. end algorithm
```

首先将当前序列所需总时间 T 置 0。对于输入的物品装填序列,对当前物品(初始当前物品为顺序中的第一个)进行判断。若可以装下当前物品,则将背包当前容量 S 减去该物品的重量,装入该物品;若不能装下,则取出物品,进行迭代:取已装入物品中剩余加工时间最短的一个物品的剩余加工时间为 t_{min},遍历已装入的物品,将其剩余时间减去 t_{min},如果减去 t_{min}后为 0,则取出该物品,再将 S 加上该物品的重量;T←$T+t_{min}$。重复取出过程,直到当前物品可以装入为止,将 S 减去该物品的重量,并标记该物品为已装入,考虑下一个物品。如果至第 n 个物品时,找到当前所有物品中所需时间的最大值 t_m,则 T←$T+t_m$。最后,将物品和背包容量初始化为原始值,结束。

分析上述过程的复杂度:当装入一个物品时,假设要取出 i 个物品,确定一个要取出物品的时间复杂度是 $O(\log n)$,则取出这 i 个物品的时间复杂度为 $i\cdot O(\log n)$。依次装入 n 个物品的时间复杂度为 $\sum_{1}^{n} i\cdot O(\log n)=n\cdot O(\log n)=O(n\cdot \log n)$。

3.2 遗传算法

对于求解 KVTS 问题的遗传算法,给出如下具有问题特性的定义。

定义 1(个体) 染色体带有特征的实体。这里,一个物品装填序列即为一个个体,个体的编码即为物品的装填序列号。

定义 2(种群) 一个种群由经过基因编码的一定数目的个体组成,这里即为若干物品装填序列的集合。种群规模指一个种群中个体的数量,用 *scale* 表示,取值根据物体的个数来调整。

定义 3(适应度) 适应度和个体装填所需的总时间相关,总时间越少,适应度越好。由于总装填时间越少的个体其适应度越大,因此选择个体装填时间的倒数作为适应度。

个体 X 的适应度 f 的计算式如式(3)所示。

$$f(X)=\frac{\alpha}{SE(X,n,S)} \tag{3}$$

其中,S 为背包总容量;α 是放大系数,为了防止数值太小,对适应度进行适度放大,本文实验中 α 设为 10,也可根据需要调整。

定义 4(选择,select) 根据选择策略产生新种群,以大概率选择适应度高的个体放在新种群靠前的位置。

选择策略采用赌轮选择法。设想群体全部个体的适应性分数由一张饼图表示,块的大小与染色体的适应性分数成正比。为了选取一个染色体,须旋转这个轮盘,当轮盘停止时,指针停止的位置对应的染色体即是选择的染色体。

定义 5(交叉,crossover)　产生两个随机数,根据这两个数将两个要交叉的染色体分成 3 段,如图 3 所示,图中数字代表物品装填的序列号。

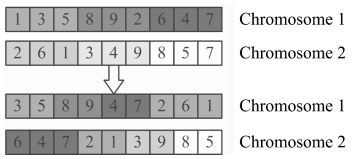

图 3　交叉操作

Fig. 3　Crossover

第一个染色体分为 3 段(1,3,5;8,9,2;6,4,7),第二个染色体分为 3 段(2,6,1;3,4,9;8,5,7)。这里的交叉是把第一个染色体的第 3 段(6,4,7)换至第二个染色体的最前位置,由于原 2 号染色体中含有个体 6,4,7,因此需要删去这几个重复的个体后更新 2 号染色体;把第二个染色体的第 1 段(2,6,1)换至第一个染色体的最后位置,去除掉原 1 号染色体中的重复部分(2,6,1)后,更新 1 号染色体。在群体更新时,将两个染色体进行交叉的概率称为交叉概率,将其预设为 0.618。

定义 6(变异,mutation)　这里的变异操作针对单个个体,对其物品序列中的一定个数(根据物品数量随机产生)的物品对的位置两两交换,从而产生新的个体。在群体更新时,将没有进行交叉操作的染色体发生变异的概率称为变异概率,将其预设为 0.09。

定义 7(群体更新)　按照预设的条件,对整个种群进行交叉变异操作,完成对当前种群的一次群体更新。每次群体更新后,将代数加 1,总更新代数有限制。

根据种群规模产生原始种群,记为 *oldPopulation*,其中第一个个体为动态规划算法得到的物品装填序列,用于保证遗传算法的初始解不会太差。其他个体随机生成,用于确保种群具有多样性。然后根据种群中每个个体的适应度 f 计算 *oldPopulation* 中每个个体的累积概率,用数组 Pr 保存,作为赌轮选择法的一部分。总适应度记为 $\sum_{i=1}^{m} f(i)$,这里 $f[i]$ 为第 i 个个体的适应度,m 是种群规模。

每个个体累计概率的计算式如式(4)所示。

$$Pr[i]=\begin{cases}\dfrac{f[1]}{\sum_{1}^{m} f(i)}, & i=1\\ \dfrac{f[i]}{\sum_{1}^{m} f(i)}+Pr[i-1], & 1<i\leqslant m\end{cases} \tag{4}$$

当更新次数达到最大代数时,输出最优时间和它对应的物品装填序列。当更新次数没有超过最大代数时,进行群体更新。群体更新时,首先产生新的种群,记为 *newPopulation*,第一个个体是 *oldPopulation* 中装填时间结果最好的个体,剩下的顺序用赌轮法确定,随机产生一个概率值,在 P_i 中找到第一个大于或等于该随机概率值的结果,并把对应的个体顺序加入 *newPopulation*,直到新种群的个体数量达到上限(*oldPopulation* 中的个体可能不会被选入 *newPopulation*)。群体更新时,以两个个体(即序列)为单位依次进行操作,首先产生一个随机数,如果这个随机数不超过交叉的概率值,那么把这两个进行交叉。否则,再产生一个随机数,如果不超过变异的概率,将第一个个体进行变异;再产生一个随机数,如果不超过变异的概率,将第二个个体进行变异。然后将 *oldPopulation* 更新为 *newPopulation* 的结果,重新计算种群的适应度和个体的累积概率,准备进行新一轮的群体更新。

整个遗传算法如算法 4 所示。

算法 4　遗传算法

输入:背包承重 S,物品数量 n,物品的集合 X

输出:在进化代数限制内的最优时间 T_{best}

```
1. initialize(oldpopulation);
2. T_best←T_oldpopulation[1] /*将 oldpopulation 中第一个个体的装填时间作为 T_best 的初始值*/
3. calculate f;/*计算每个个体的适应度*/
4. calculate Pr;/*计算每个个体的累积概率*/
5. l← 0/*当前的进化代数*/
6. while l <Max_Gen do /*计算累积概率*/
7.    T←selectbest(oldpopulation);/*找到最佳个体,该个体的装填时间为 T*/
8.    if T_best>T then
9.       T_best← T;
10.   endif
11.   select(oldpopulation,newpolulation)/*赌轮选择法生成新种群*/
12.   for k←1 to scale,k←k+2 do
13.      r←random(0,1);/*产生随机数,范围为 0～1*/
14.      if r<Pc then
15.         Crossover(k,k+1);
16.      endif
17.      r←random(0,1);
18.      if r<Pm then
19.         mutation(k);
20.      endif
21.      r←random(0,1);
22.   if r<Pm then
23.      mutation(k+1);
24.   endif
25.   oldpopulation←newpolulation;
26.   l←l+1;
27. end while
28. return T_best;
29. end algorithm
```

本节介绍了考虑时间因素的 0-1 背包问题的遗传算法。以物品装填序列作为个体,随机产生原始种群,并将动态规划算法得到的结果作为初始种群中的一个个体,然后选择原始种群中最好的个体作为新种群的第一个个体并进行记录,其他个体用赌轮选择法产生,生成新种群。对新种群进行交叉变异操作并将其作为一次群体更新,用群体更新完毕后的新种群替代原始种群,再按照同样的规则进行新一轮的群体更新,直到更新代数超出限制,同时记录整个群体更新过程中出

现的最优解作为求解该问题的最优解。

4 实验与结果

4.1 实验环境

在笔记本电脑上完成算法的实验测试，平台的硬件与环境如表 1 所列。

表 1 实验环境说明

Table 1 Experimental environment description

Type	specific information
OS	Microsoft Windows 10 Pro version 64bit
CPU	Intel® Core™ i7-3610QM CPU @ 2.30GHz
Memory	8GB
Disk	OCZ-VERTEX4 128GB
IDE	Codeblocks 13.12

4.2 测试数据及结果汇总

主要通过 3 组设计的算例来完成对算法的实验测试。

第一组算例基于文献[25]，这是一组经典的矩形 Packing 问题的算例，我们将矩形块的宽视为物品的重量，将矩形块的高视为放置物品所需的时间，从而将其转化为本问题的算例。

测试结果汇总于表 2。这组算例由于物品数量较多，因此没有测试完备算法。本组算例中物品的大小以及所需的放置时间较为接近，且物品重量相对于背包承重而言较小(作为矩形 Packing 问题的算例时，面积利用率接近 100%)，动态规划已经能得到足够好的结果，遗传算法并没有带来足够大的提升。遗传算法在大部分情况下能得到比动态规划算法稍好的结果，但需要花费的时间远多于动态规划算法的时间。

表 2 第一组算例的测试结果

Table 2 Test results of the first set of instances

Instance	n	Backpack capacity	IDP		Genetic algorithm	
			Result	Running time/s	Result	Running time/s
C1.1	16	20	21	0.016	20	40.037
C1.2	17	20	22	0.016	21	42.062
C1.3	16	20	21	0.016	20	35.142
C2.1	25	40	16	0.016	16	45.017
C2.2	25	40	16	0.016	16	47.017
C2.3	25	40	15	0.016	15	44.017
C3.1	28	60	31	0.016	31	57.017
C3.2	29	60	32	0.016	31	50.297
C3.3	28	60	32	0.016	31	45.031
C4.1	49	60	62	0.016	62	74.016
C4.2	49	60	64	0.016	63	77.707
C4.3	49	60	62	0.016	62	86.016
C5.1	73	60	92	0.016	91	190.327
C5.2	73	60	93	0.016	92	211.169
C5.3	73	60	92	0.018	91	214.971
C6.1	97	80	122	0.016	121	277.678
C6.2	97	80	124	0.016	122	313.698
C6.3	97	80	123	0.016	122	276.988
C7.1	196	160	241	0.028	241	620.059
C7.2	197	160	246	0.028	243	573.664
C7.3	196	160	242	0.027	242	555.055

第二组算例为随机生成，同构性较弱，测试结果如表 3 所列。该组为小规模算例测试，仅用于测试完备算法的可行性。可以看到，随着物品数量的增加，完备算法消耗的时间呈阶乘级别增长，物品数量超过 15 时计算时间过长，因此物品数量的上限取为 15。注意，15 的阶乘已经是万亿级别，说明 BnB 算法已经进行了大量的剪枝操作。实验发现当深度较大时才能剪枝，在后续的研究中，可考虑改进对节点下界进行估计的方法来改进剪枝效果，以进一步缩短运行时间。

表 3 第二组算例

Table 3 Test results of the second set of instances

Instance	n	BnB		IDP		Genetic algorithm	
		Result	Running time/s	Result	Running time/s	Result	Running time/s
1	13	67	30.754	67	0.016	67	14.016
2	14	97	386.653	102	0.016	97	17.816
3	15	100	2011.748	107	0.016	100	23.016
4	15	123	2682.125	131	0.019	123	20.036
5	15	126	2757.256	129	0.022	126	24.031

可以看到，随着测试数据异构性的增强，遗传算法在多数时候都可以找到比动态规划算法更优的解。同时，因为物品数量较少，遗传算法得到的结果与 BnB 相同，达到了理论最优解，但运行时间远远少于 BnB，说明遗传算法在这些算例上的实现效果较好。

第三组算例为随机生成，同构性较弱，规模较大，结果如表 4 所列。这里将贪心比定义为动态规划的最优结果与遗传算法得到的最优结果的比值，比值越小，说明遗传算法相比动态规划算法提升越大。

表 4 第三组算例的测试结果

Table 4 Test results of the third set of instances

Instance	n	IDP		Genetic algorithm		
		Result	Running time/s	Result	Running time/s	Greedy ratio
6	100	5728	0.021	5340	143.284	0.932
7	125	4558	0.031	4179	216.991	0.917
8	150	11301	0.033	10958	250.903	0.970
9	175	8582	0.042	8176	424.995	0.952
10	200	12431	0.053	11663	551.106	0.938
11	225	16049	0.069	14872	769.470	0.927
12	250	17415	0.075	16270	808.373	0.934
13	275	22252	0.130	21564	829.119	0.969
14	300	26499	0.126	25464	1878.752	0.961
15	325	32978	0.185	30719	1448.342	0.932
Average	213	15779	0.077	14921	732.134	0.943

该组测试数据全部随机生成且规模较大。从测试结果可以看出，由于动态规划算法的时间复杂度很低，因此在算例较大时依然可以在非常短的时间内得到结果。

当算例异构性增强、规模较大时，遗传算法是从一个初始种群开始，规模较大，且交叉、变异操作使得遗传算法很难陷入局部最优解，可以高效地搜索到很大的解空间，比单纯的动态规划算法优秀得多。与动态规划算法相比，遗传算法得到的结果平均约有 6%的提升，最大提升约为 9%。因此，在算例完全随机的情况下，动态规划算法相对于理论最优解的相对误差至少是 9%。

通过对 3 组算例的分析比较可以得出如下结论：动态规划算法可以在很短的时间内得到较满意的结果，在物品时间相差较少时，结果很接近历史最优解。完备算法可以给出问题的理论最优解，但由于搜索空间巨大，只能在可接受的时间里计算出最多包含 15 个物品的小规模算例的最优装填结果。

遗传算法在大多数算例上都可以得到比动态规划算法更好的解，在大规模随机算例中表现优异，结果相对于动态规划算法有 6%～7%的改进。

结束语　文中提出了考虑时间因素的 0-1 背包问题(KVTS)这一经典 NP 难度问题——0-1 背包问题的拓展问题，并对该问题提出了 3 种求解算法：迭代动态规划算法、基于分枝限界法的完备算法和遗传进化算法；同时设计生成了 3 组 KVTS 问题的算例，并对所提出的 3 种算法进行了测试对比。实验结果表明，迭代动态规划算法可以在很短的时间内得到较满意的结果；基于分枝限界的完备算法可以计算出小规模算例的理论最优解；对于较大规模的随机算例，遗传算法相比迭代动态规划算法在结果上可以改进约 6%～7%。

参 考 文 献

[1] STEINBERG A. A Strip-Packing Algorithm with Absolute Performance Bound 2[J]. SIAM Journal on Computing, 1997, 26(2):401-409.

[2] ZHU P, HE K, CAO W G, et al. A Caving-degree based Greedy Scheduling Algorithm for the Three-dimensional Space-Time Optimization Problem[J]. Journal of Frontiers of Computer Science & Technology, 2016, 10(8):1051-1062. (in Chinese)
朱鹏，何琨，曹伟刚，等. 基于穴度的三维时空优化问题的贪心调度算法[J]. 计算机科学与探索，2016，10(8)：1051-1062.

[3] HUANG W Q, HE K. On the Weak Computability of a Four-dimensional Orthogonal Packing and Time Scheduling Problem [J]. Theoretical Computer Science, 2013, 501(3):1-10.

[4] HUANG W Q, HE K. An Optimal Time Scheduling Problem on Cuboids Packing over Four-Dimensional Space-Time and Its Computability Proof[J]. Chinese Journal of Computers, 2013, 36(9):1880-1888. (in Chinese)
黄文奇，何琨. 四维时空高效利用的装箱调度问题及其可计算性证明[J]. 计算机学报，2013，36(9)：1880-1888.

[5] HUANG W Q, HE K. Optimal Time Scheduling of Three-Dimensional Space Packing[J]. Journal of Huazhong University of Science and Technology (Natural Science Edition), 2010, 38(12):102-104. (in Chinese)
黄文奇，何琨. 三维装箱工作的优化调度问题[J]. 华中科技大学学报(自然科学版)，2010，38(12)：102-104.

[6] MERKLE R C, HELLMAN M E. Hiding Information and Signatures in Trapdoor Knapsacks[J]. IEEE Transactions on Information Theory, 1978, 24(5):525-530.

[7] CONNOLLY D. Knapsack Problems: Algorithms and Computer Implementations[J]. Journal of the Operational Research Society, 1991, 42(6):513-513.

[8] HU J S, CHEN G L, GUO G C. Solving the 0/1 Knapsack Problem on Quantum Computer[J]. Chinese Journal of Computers, 1999, 22(12):1314-1316.

[9] CORMEN T H, LEISERSON C E. Introduction to Algorithm [M]. Beijing: China Machine Press, 2013.

[10] IBARRA O H, KIM C E. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems[J]. Journal of the Acm, 1975, 22(4):463-468.

[11] DU D Z, KERI K O, HU X D. Design and Analysis of Approximation Algorithms[M]. New York: Springer, 2012.

[12] YANG Z X, YONG Z Z, YU M, et al. Improved Genetic Algorithm for Solving Knapsack Problem[J]. Journal of Shenzhen University (Science & Engineering), 2006, 23(2):128-132. (in Chinese)
杨泽星，雍正正，俞敏，等. 解决背包问题的改进遗传算法[J]. 深圳大学学报(理工版)，2006，23(2)：128-132.

[13] DAREHMIRAKI M, NEHI H M. Molecular Solution to the 0-1 Knapsack Problem Based on DNA Computing[J]. Applied Mathematics & Computation, 2007, 187(2):1033-1037.

[14] MA L, WANG L D. Ant Colony Optimization Algorithm for Knapsack Problem[J]. Journal of Computer Applications, 2001, 21(8):4-5. (in Chinese)
马良，王龙德. 背包问题的蚂蚁优化算法[J]. 计算机应用，2001，21(8)：4-5.

[15] IMAHORI S, YAGIURA M, IBARAKI T. Local Search Algorithms for the Rectangle Packing Problem with General Spatial Costs. Math Program[J]. Mathematical Programming, 2004, 167(1):48-67.

[16] GEHRING H, BORTFELDT A. A Parallel Genetic Algorithm for Solving the Container Loading Problem[J]. International Transactions in Operational Research, 2002, 9(9):497-511.

[17] LEUNG T W, YUNG C H, TROUTT M D. Applications of Genetic Search and Simulated Annealing to the Two-Dimensional Non-guillotine Cutting Stock Problem[J]. Computers & Industrial Engineering, 2001, 40(3):201-214.

[18] CHAN H H, MARKOV I L. Practical Slicing and Non-slicing-Block-packing without Simulated Annealing[C]// Proceedings of the 14th ACM Great Lakes symposium on VLSI. ACM, 2004: 282-287.

[19] CHEN M, HUANG W. A Two-level Search Algorithm for 2D Rectangular Packing Problem[J]. Computers & Industrial Engineering, 2007, 53(1):123-136.

[20] BAKER B S, COFFMAN E G, RIVEST R L. Orthogonal Packings in Two Dimensions[J]. Siam Journal on Computing, 1980, 9(4):846-855.

[21] CHAZELLE B. The Bottomn-Left Bin-Packing Heuristic: An Efficient Implementation[J]. IEEE Transactions on Computers, 1983, C-32(8):697-707.

[22] HOPPER E. Two-Dimensional Packing Utilising Evolutionary Algorithms and Other MetaHeuristic Methods[D]. Wales: University of Wales Cardiff, 2000.

[23] ZHANG D, LIU Y, CHEN S, et al. A Meta-heuristic Algorithm for the Strip Rectangular Packing Problem[J]. Lecture Notes in Computer Science, 2005, 3612(8):437-437.

[24] HOLLAND J H. Adaptation in Natural and Artificial Systems [M]. Massachusetts: MIT Press, 1992.

[25] HOPPER E, TURTON B C H. An Empirical Investigation of Meta-heuristic and Heuristic Algorithms for a 2D Packing Problem[J]. European Journal of Operational Research, 2001, 128(1):34-57.