2025 秋 -《算法设计与分析》
回溯与分支限界算法分析实验报告

实 验 时 间

2026.1.19

《本科实验报告》填写说明

实验报告内容编排应符合以下要求：

（1）采用 A4（21cm×29.7cm）白色复印纸，单面黑字。上下左右各侧的页边距均为 3cm；缺省文档网格：字号为小 4 号，中文为宋体，英文和阿拉伯数字为 Times New Roman，每页 30 行，每行 36 字；页脚距边界为 2.5cm，页码置于页脚、居中，采用小 5 号阿拉伯数字从 1 开始连续编排，封面不编页码。

（2）报告正文最多可设四级标题，字体均为黑体，第一级标题字号为 4 号，其余各级标题为小 4 号；标题序号第一级用“一、”、“二、”……，第二级用“（一）”、“（二）”……，第三级用“1.”、“2.”……，第四级用“（1）”、“（2）”……，分别按序连续编排。

（3）正文插图、表格中的文字字号均为 5 号。

目录

图目录

1 实验介绍

回溯法（Backtracking）和分支限界法（Branch and Bound）是求解组合优化问题的两种重要算法。回溯法通过深度优先搜索状态空间树，利用剪枝函数避免无效搜索；分支限界法则常采用广度优先或最佳优先策略，利用代价函数（Bound）计算结点的上界（或下界），以剪除不可能产生最优解的分支，从而加速搜索。本实验旨在通过完全背包问题和多重背包问题，深入理解这两种算法的原理，特别是代价函数的设计对算法性能的影响，并掌握蒙特卡洛方法在估算搜索树规模中的应用。

2 实验内容

本实验主要包含以下内容：

1. 针对完全背包问题，实现回溯法与分支限界算法。
2. 利用蒙特卡洛方法对搜索树的分支数量进行估计。
3. 分析分支限界法中代价函数的准确性，通过与真实值（由动态规划求得）对比，分析不同层级和不同输入规模下的近似效果。
4. 设计并对比两种不同的代价函数（朴素界与分数背包界），分析其剪枝效果与计算开销。
5. （附加）针对多重背包问题，实现分支限界算法，并对比不同代价函数的性能。

3 实验要求

具体要求如下：

1. 以物品种类数 n 为输入规模，随机生成测试样本。
2. 统计不同算法的运行时间、访问结点数。
3. 使用 Python 绘制数据图表，展示蒙特卡洛估计结果、代价函数近似比、以及不同算法的性能对比。
4. 分析实验结果，验证理论分析。

4 实验步骤

4.1 算法设计

4.1.1 完全背包问题的分支限界法

完全背包问题允许每种物品选择无限次。在分支限界法中，我们构建状态空间树。为了便于剪枝，我们将物品按价值密度（$\frac{v_i}{w_i}$）降序排列。

```
struct Item {
    int id; int weight; int value;
    double density; int limit;
    Item(int id, int w, int v, int l = -1) : id(id), weight(w), value(v),
limit(l) {
        density = (double)v / w;
    }
};
```

```
bool compareItems(const Item& a, const Item& b) {
    return a.density > b.density;
}
```

每个结点包含当前价值 V_{cur}、当前重量 W_{cur} 和当前考虑的物品层级 level。我们使用二叉分支策略：

1. 左孩子：选择当前物品一件，状态更新为 $(\text{level}, W_{\text{cur}} + w_i, V_{\text{cur}} + v_i)$，前提是未超重。
2. 右孩子：不再选择当前物品，转而考虑下一件物品，状态更新为 $(\text{level} + 1, W_{\text{cur}}, V_{\text{cur}})$。

为了进行剪枝，我们需要计算当前结点的价值上界（Upper Bound, UB）。如果 UB $\leq$ current_best，则剪枝。我们实现了两种代价函数：

1. 朴素界 **(Simple Bound)**：假设剩余容量全部以全局最大单位价值填充。

$$\text{UB} = V_{\text{cur}} + (W - W_{\text{cur}}) \text{ Times New Roman } \max\left(\frac{v_i}{w_i}\right)$$

   该界计算简单，但较为松弛。

2. 分数背包界 **(Fractional Bound)**：即标准的分支限界法上界。将剩余空间用分数背包问题的贪心解填充（即优先装入密度大的物品，最后一件可分割）。由于物品已排序，该界能提供更紧密的上界。

```
double bound_fractional(int level, int current_val, int rem_cap, const
vector<Item>& items) {
    double bound = current_val;
    int w = rem_cap;
    for (int i = level; i < items.size(); ++i) {
        if (w >= items[i].weight) {
            // Take as many as possible (for complete knapsack fractional)
            bound += (double)w * items[i].density;
            return bound;
        }
    }
    return bound;
}
```

4.1.2 蒙特卡洛方法估算搜索树规模

对于大规模问题，直接遍历搜索树是不现实的。蒙特卡洛方法通过随机采样路径来估算树的结点总数。设路径上第 i 层结点的度数为 m_i，则该路径代表的树规模估计值为：

$$N = 1 + m_0 + m_0 m_1 + m_0 m_1 m_2 + \ldots$$

```
long long monte_carlo_estimate(const vector<Item>& items, int capacity, int
samples = 1000) {
    long long total_nodes = 0;
    for (int k = 0; k < samples; ++k) {
        long long current_multiplier = 1;
        // ... (traversal logic) ...
        int branching_factor = moves.size();
        total_nodes += current_multiplier;
        current_multiplier *= branching_factor;
        // ...
    }
```

```
    return total_nodes / samples;
}
```

通过多次采样取平均值，可得到搜索树规模的无偏估计。在完全背包问题中，由于分支因子变化较大（取决于剩余容量），该方法能有效预估问题难度。

4.1.3 多重背包问题的分支限界法

多重背包问题中，每种物品的数量有限制 k_i。算法结构与完全背包类似，但在分支时需考虑物品数量限制。此处同样对比了两种代价函数：

1. 松弛界 **(Loose Bound)**：忽略数量限制，视为完全背包求分数界。
2. 紧致界 **(Tight Bound)**：考虑数量限制求解分数背包问题。即在贪心填充时，不仅受容量限制，也受物品数量 k_i 限制。

```
double bound_mk_tight(int level, int current_val, int rem_cap, const
vector<Item>& items) {
    double bound = current_val;
    int w = rem_cap;
    for (int i = level; i < items.size(); ++i) {
        if (items[i].weight == 0) continue;
        int can_take_weight = items[i].limit * items[i].weight;
        if (w >= can_take_weight) {
            w -= can_take_weight;
            bound += items[i].value * items[i].limit;
        } else {
            bound += (double)w * items[i].density;
            return bound;
        }
    }
    return bound;
}
```

4.2 实验环境

- 操作系统：Linux
- 编程语言：C++ (G++)
- 数据分析：Python (Pandas, Seaborn, Matplotlib)
- 硬件环境：标准 PC

5 实验结果与分析

5.1 蒙特卡洛搜索树规模估计

图 1 展示了随物品种类数 n 增加，完全背包问题搜索树结点数的蒙特卡洛估计值（对数坐标）。

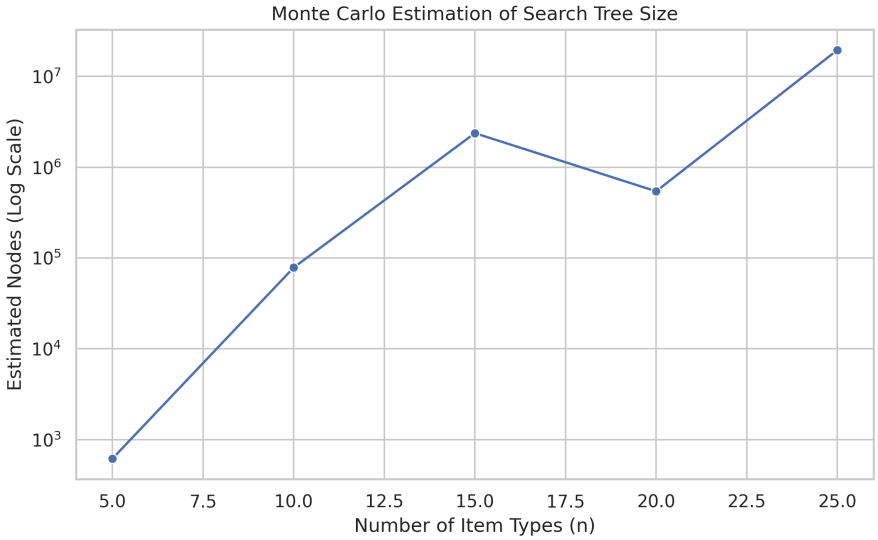

Figure 1: 搜索树规模的蒙特卡洛估计

结果表明, 搜索树规模随 n 呈指数级增长。蒙特卡洛方法能够快速给出问题规模的数量级估计, 对于判断是否能在有限时间内求出精确解具有指导意义。对于整数背包问题，当 n 较大时，建议先使用蒙特卡洛方法预估，若规模过大则应考虑近似算法或启发式搜索。

5.2 代价函数准确性分析

为了评估代价函数（上界）的质量，我们记录了搜索过程中各结点的上界值与该状态下的真实最优值（通过动态规划预先计算得到）的比值。比值越接近 1，说明上界越紧致。

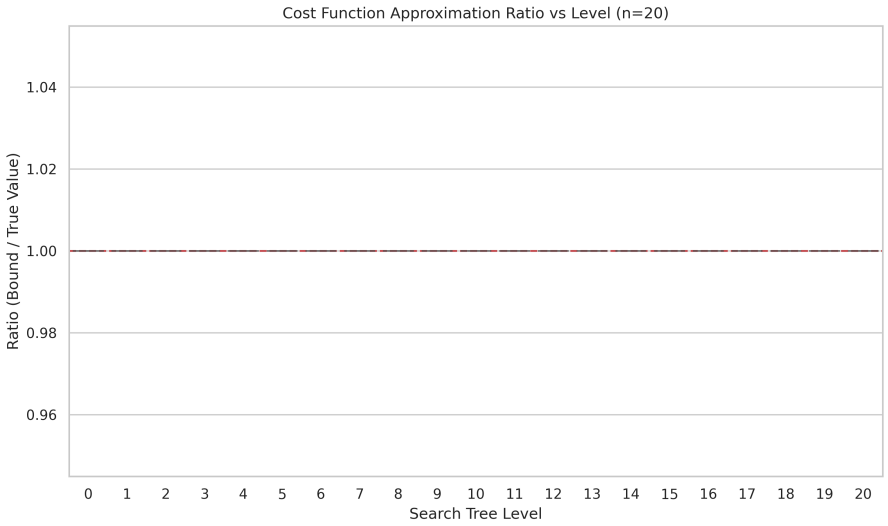

Figure 2: 不同层级下代价函数的近似比 (n=20)

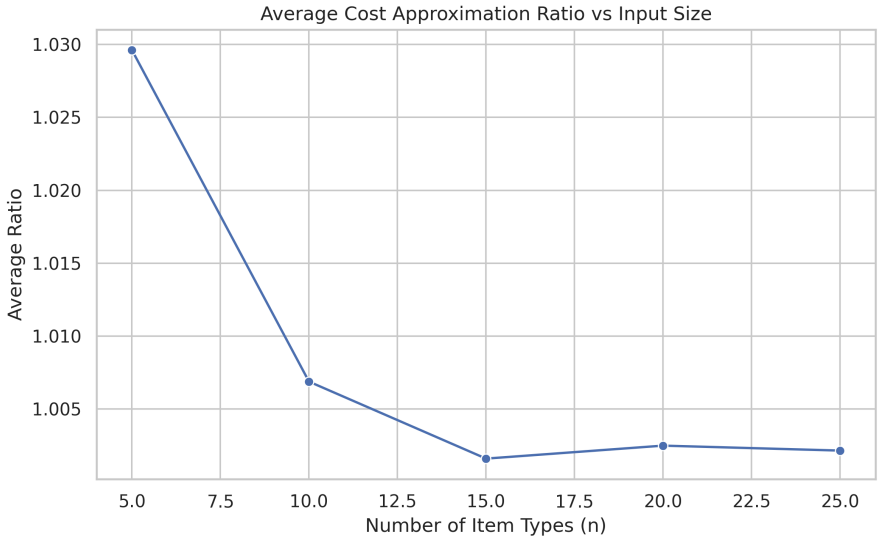

Figure 3: 平均近似比随输入规模 n 的变化

从图 2 可以看出，随着搜索深度的增加（Level 增大），剩余问题规模变小，代价函数的近似比逐渐趋向于 1，说明上界越来越精确。这是符合预期的，因为随着物品确定的越多，不确定性越小。 图 3 展示了输入规模 n 对平均近似比的影响。通常情况下，平均近似比相对稳定，不会随 n 剧烈波动，这表明分数背包界具有良好的鲁棒性。

5.3 不同代价函数的性能对比

我们对比了“分数背包界 (Fractional)”与“朴素界 (Simple)”在完全背包问题上的性能。

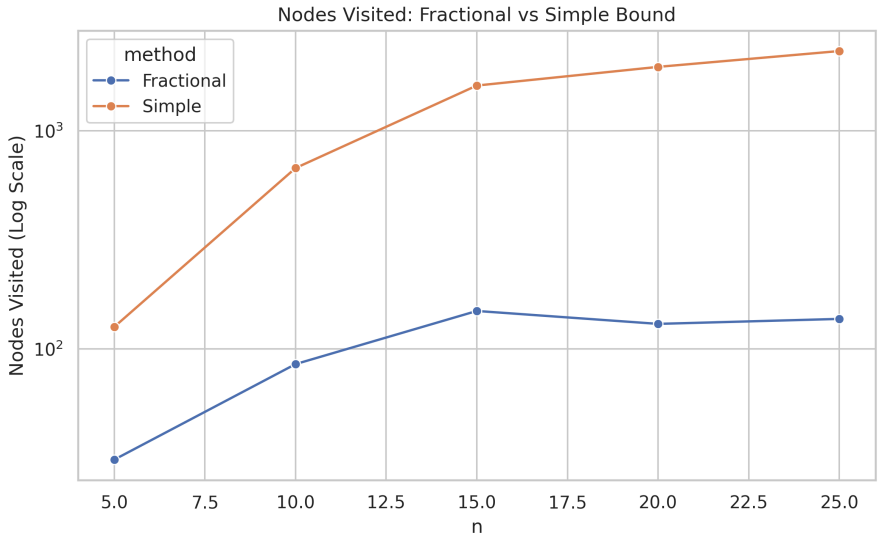

Figure 4: 不同代价函数下的访问结点数对比

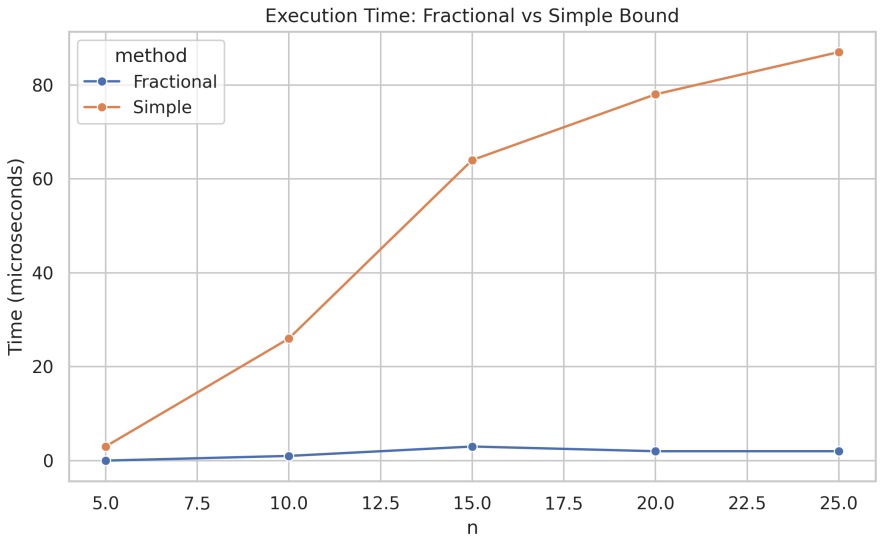

Figure 5: 不同代价函数下的运行时间对比

实验结果显著：

1. 剪枝效果：分数背包界（Fractional）的访问结点数远少于朴素界（Simple），常常相差数个数量级（注意图 4 为对数坐标）。这是因为分数背包界提供了更紧的上界，能更早地剪除无效分支。
2. 运行时间：尽管分数背包界的计算复杂度略高于朴素界（需要遍历剩余物品，而朴素界仅需常数/一次乘法），但由于其极强的剪枝能力，总运行时间反而大幅降低。

这说明在分支限界法中，设计一个计算稍复杂但更紧致的代价函数通常是值得的。

6 附加：多重背包问题分析

在多重背包问题中，我们对比了考虑物品数量限制的“紧致界 (TightBound)”与忽略数量限制的“松弛界 (LooseBound)”。

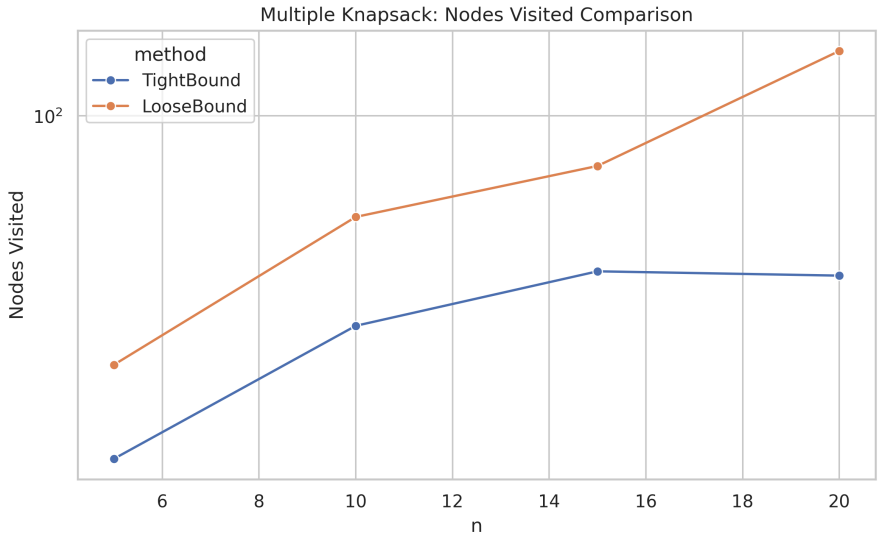

Figure 6: 多重背包：不同代价函数的结点数对比

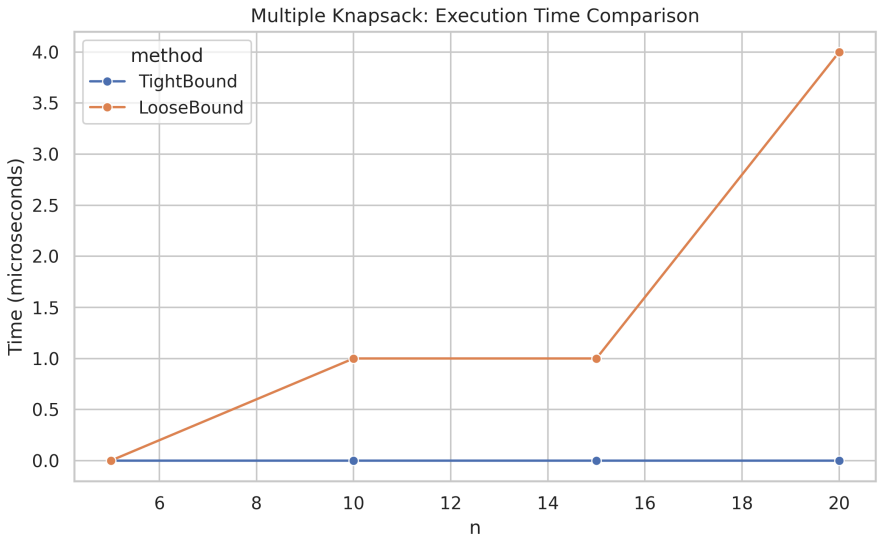

Figure 7: 多重背包：不同代价函数的运行时间对比

结果显示，紧致界（TightBound）在性能上优于松弛界。因为忽略数量限制会导致上界过大，无法有效剪除那些虽然总重量满足但单种物品数量超标的分支。通过在代价函数中精确建模约束条件，可以显著提高算法效率。

7 实验总结

本实验通过实现和分析完全背包及多重背包问题的分支限界算法，得出以下结论：

1. 代价函数的重要性：代价函数的紧致程度直接决定了分支限界法的剪枝效率。更紧的界（如分数背包界）虽然单次计算开销稍大，但能指数级减少搜索空间，从而获得更好的总性能。
2. 蒙特卡洛方法的实用性：该方法能有效评估大规模组合优化问题的解空间大小，为算法选择提供依据。
3. 真实值对比分析：通过与 DP 得到的真实值对比，验证了分支限界法随着搜索深度增加，对问题最优解的估计越来越准确的特性。