浙江大學
ZHEJIANG UNIVERSITY

姓名 _____

学号 _____

院所 _____

2025 年 11 月 30 日

Contents

1 项目描述 (Project Description)

本项目旨在解决 **3-Partition Problem**（三路划分问题）。给定一个包含 N 个正整数的多重集合 $S = \{a_1, a_2, \ldots, a_N\}$，我们需要判断是否可以将 S 划分为三个互不相交的子集 S_1, S_2, S_3，使得这三个子集的元素之和相等。

形式化地，令总和 $Sum = \sum_{i=1}^{N} a_i$。我们需要找到划分满足：

$$\sum_{x \in S_1} x = \sum_{y \in S_2} y = \sum_{z \in S_3} z = \text{Target} \tag{1}$$

其中 Target $= Sum/3$。如果 Sum 不能被 3 整除，则直接判定为无解。如果存在这样的划分，算法需要输出具体的子集划分方案；否则输出 “no”。

2 算法描述 (Description of Algorithms)

针对该问题，我们设计并实现了三种算法：基于剪枝的深度优先搜索 (DFS)、伪多项式时间的动态规划 (DP) 以及启发式的模拟退火 (Simulated Annealing)。

2.1 算法一：带剪枝的深度优先搜索 (DFS)

DFS 是一种精确算法，通过递归地构建搜索树来遍历所有可能的状态。算法的核心是一个递归函数 `DFS(index, bucket_sums)`，其中 `index` 表示当前正在处理的数字下标，`bucket_sums` 维护当前三个桶的累加和。为了应对最坏情况下 $O(3^N)$ 的指数级搜索空间，我们使用了两项剪枝策略：

1. **降序排序 (Sort Descending)**: 在搜索开始前，我们将输入数组 S 按从大到小的顺序排序。
   - *原理*: 优先处理数值较大的元素可以更快地减少剩余桶的可用容量。较大的元素灵活性较差（更难放入剩余空间小的桶），因此能更早地触发容量溢出条件，从而实现“快速失败 (Fail-Fast)”，显著减少无效递归的深度。
2. **等效性剪枝 (Symmetry Breaking)**: 这是本算法的核心优化，用于消除搜索树中的同构子树。
   - *原理*: 初始时，三个空桶在数学上是完全等价的（即划分 $\{A, B, C\}$ 和 $\{B, A, C\}$ 是同一种解）。如果当前数字 a_{index} 尝试放入第一个空桶后导致后续搜索失败，那么尝试将其放入第二个或第三个空桶也必然会得到相同的结果。
   - *实现*: 在递归过程中，如果当前桶是空的（Sum=0），且将当前数字放入该桶导致了回溯（即该分支无解），我们强制停止尝试后续所有的空桶。这有效地定义了桶的“填入顺序”，将搜索空间减少了约 3! 倍。

2.2 算法二：动态规划 (DP)

我们将该问题建模为多维 0/1 背包问题的变种。由于我们需要将数字精确地分配到三个桶中，我们只需追踪前两个桶的状态，第三个桶的状态由总和守恒定律隐含确定。

- **状态定义**: 令 $dp[i][j]$ 为布尔值，表示是否可以从前 k 个物品中选出部分物品填满桶 1 至容量 i，同时填满桶 2 至容量 j。如果 $dp[i][j]$ 为真，且 $TotalSum\_k - i - j \leq Target$，则状态合法。

- **状态转移方程**: 对于第 k 个物品（重量为 v），我们遍历所有可能的 (i, j) 状态。新的状态 dp_{new} 可以由上一轮状态 dp_{old} 转移而来：

$$dp[i][j] = \underbrace{dp[i][j]}_{\text{放入桶 3}} \vee \underbrace{dp[i-v][j]}_{\text{放入桶 1}} \vee \underbrace{dp[i][j-v]}_{\text{放入桶 2}} \tag{2}$$

  边界条件为 $dp[0][0] = \text{True}$。

- **路径记录与回溯**: 为了输出具体方案，我们维护一个三维数组 $path[k][i][j] \in \{1, 2, 3\}$。
  - 当我们将第 k 个物品放入桶 1 并更新状态 (i, j) 时，记录 $path[k][i][j] = 1$。
  - 算法结束后，若 $dp[Target][Target]$ 为真，我们从 $(N, Target, Target)$ 开始逆向回溯，根据 $path$ 数组还原每个物品的归属。

2.3 算法三：模拟退火 (Simulated Annealing)

为了解决大规模输入（$N = 1000$）或 Bonus 部分的高维划分问题（Exact Algorithms 会因时间或内存耗尽而失效），我们实现了模拟退火这一基于概率的启发式算法。

1. **初始化 (Initialization)**: 随机将 N 个数分配到 3 个桶中，作为初始解 S_0。

2. **能量函数 (Cost Function)**: 我们定义系统的"能量"为当前状态的不平衡度。目标是寻找全局最小能量 $E = 0$。

$$E(S) = \sum_{m=1}^{3} |Sum_m - \text{Target}| \tag{3}$$

3. **邻域操作 (Neighbor Generation)**: 在每次迭代中，我们通过微扰当前状态产生新状态 S'。采用了两种策略：
   - *Move*: 随机选择一个数字，将其从当前桶移动到另一个随机桶。
   - *Swap*: 随机选择两个分别位于不同桶的数字，交换它们的位置。

4. **Metropolis 准则**: 令 $\Delta E = E(S') - E(S)$。
   - 若 $\Delta E < 0$（新状态更优），则无条件接受新状态。
   - 若 $\Delta E \geq 0$（新状态更差），则以概率 $P = \exp(-\Delta E/T)$ 接受新状态。其中 T 为当前温度。这一机制允许算法在早期"爬山"，跳出局部最优解。

5. **卡时重启 (Time-Limited Restart)**: 由于 SA 是概率算法，单次运行可能无法收敛。我们引入了重启机制：如果在降温结束时未找到解 $(E > 0)$ 且总运行时间未超过 0.9 秒，则重置温度和随机种子，从头开始新一轮退火。这在有限时间内最大化了找到解的概率。

3 时间复杂度的理论分析 (Theoretical Analysis)

3.1 问题复杂性归类

- **3-Partition Problem** 是一个 **强 NP-完全 (Strongly NP-Complete)** 问题。
- 强 NP-完全性意味着即使输入数字的大小（Value）受到多项式的限制，问题依然是 NP-完全的。这也暗示了除非 $P = NP$，否则该问题不存在 **全多项式时间近似方案 (FPTAS)**。
- **结论**：不存在以输入数据的位长（Bit Length）为多项式时间的精确算法。

3.2 算法一：深度优先搜索 (DFS)

- **最坏时间复杂度**: $O(3^N)$
- **详细分析**: DFS 算法本质上是在遍历一颗三叉树。对于集合中的 N 个元素，每一个元素都有 3 种可能的去向（桶 1、桶 2 或桶 3）。因此，未经剪枝的搜索空间大小为 $3 \times 3 \times \cdots \times 3 = 3^N$。
- **剪枝的影响**: 虽然我们在实现中加入了*降序排序*和*等效性剪枝*，这在处理随机生成的数据时极大地修剪了搜索树（使得平均分支因子远小于 3），但在最坏情况下（例如精心构造的对抗性数据，或者所有数字都相等且无解时），剪枝可能失效，算法仍需遍历大部分状态空间。因此，其理论上界依然是指数级的。

3.3 算法二：动态规划 (DP)

- **时间复杂度**: $O(N \cdot \text{Target}^2)$，其中 $\text{Target} = \text{Sum}/3$。
- **空间复杂度**: $O(N \cdot \text{Target}^2)$。
- **伪多项式时间 (Pseudo-polynomial Time)**：公式中包含 N 和 Target 的多项式，似乎是高效的。然而，在计算复杂性理论中，算法的时间复杂度是相对于输入规模（Input Size）而言的，而输入规模是按数据的二进制位长计算的。
  - 假设输入数字的最大值为 M，则输入该数字所需的比特数为 $L = \log_2 M$。
  - 算法的运行时间与数值 M（即 Target）成正比，即与 2^L 成正比。
  - 因此，运行时间相对于输入规模 L 是**指数级**增长的。

  结论：当 N 很大但数值 M 很小（如 $M \leq N$）时，DP 是多项式时间的有效解法；但当数值 M 很大（如 $M = 10^{100}$）时，DP 将因内存耗尽或超时而失效。

3.4 算法三：模拟退火 (Simulated Annealing)

- **时间复杂度**: $O(K_{iter} \cdot T_{update})$
- **分析**: 模拟退火的复杂度不由输入规模 N 直接决定，而是由人为设定的冷却计划决定。
  - K_{iter}: 总迭代次数，大约为 $\log_\alpha(\frac{T_{end}}{T_{start}})$。其中 α 是降温系数。
  - T_{update}: 每次状态转移（移动或交换）并更新能量函数的代价。在优化实现中，我们可以通过增量计算（Incremental Update）在 $O(1)$ 时间内完成能量更新。

Table 1: DFS 算法运行时间统计（秒）

n	平均时间	最大时间	最小时间	方差
10	0.0080	0.0182	0.0054	5.12e-05
15	0.0081	0.0145	0.0044	2.19e-05
20	0.0078	0.0149	0.0042	1.92e-05
25	0.0064	0.0101	0.0043	6.82e-06
30	0.0921	0.1373	0.0049	0.0045
35	2.3341	9.2076	0.0043	7.41
40	3.1284	11.2849	0.0043	17.92
45	—	超时	0.0040	—
50	103.034	254.024	0.0044	12813.5

因此，总复杂度主要取决于预设的参数。在我们的实现中，为了适应在线评测系统（OJ），我们加入了硬性的时间限制（0.9 秒）。这使得算法在工程上表现为 $O(1)$ 的常数时间复杂度（相对于问题规模无限增长而言，它总是能在固定时间内停止），但代价是无法保证解的完备性，且无法证明无解的情况。

4 实验与测试分析

本实验对三种算法方法（DFS+ 剪枝、动态规划（DP）、模拟退火（SA））在三分类问题上的性能进行了系统测试，测试设计考虑了 DFS 在小规模输入上的表现, 剪枝策略的优化, 输入规模 n 与总和 sum 的变化对 DP 算法性能的影响, 以及 SA 算法在大规模输入上的稳定性

4.1 测试数据设计

本实验共设计三类测试数据，分别用于评估 DFS、DP 与模拟退火（SA）算法在不同规模与结构下的性能表现。

首先，在 DFS 部分，我们设计了两类测试数据: 一类是小规模 n 但具有较大 sum 的输入，用于考察剪枝策略在极端情况下的有效性；另一类是 n 与 sum 均较小的典型输入，用于基准测试 DFS 在常规规模下的运行性能。

对于 DP 算法，我们构造了三组测试数据。第一组固定 n 并逐步增大 sum，用于观察 DP 算法在状态空间随总和增大时的时间增长趋势。第二组固定 sum 但逐步增大 n，以分析 DP 对输入规模的敏感性。第三组则同时连续增加 n 与 sum，用于评估 DP 在大规模数据下的可计算范围，并对其经验时间复杂度进行拟合分析。

最后，在模拟退火（SA）部分，为了构造既能保证可解性、又能使 DP 难以处理的大规模测试实例 (符合 SA 的实际使用情况)，我们采用两种思路 (base construction 和 greedy fill-in) 构造出了 n 和 sum 都较大的测试集, 并让 `/sa` 在至多重启 10 次的情况下观察性能

4.2 DFS + 剪枝测试结果

在小规模 n，较大 sum 的数据下，DFS 算法在剪枝策略失效时，计算时间显著增加，部分数据甚至出现计算时间过长的现象。测试结果统计如表 1

- 当剪枝条件失效时，DFS 的搜索空间呈指数级增长，时间复杂度接近 $O(k^n)$，其中 k 为每个元素可选分类数量。
- $n = 45$ 时出现超时现象，表明剪枝失效对 DFS 的影响极大。
- 对于小规模 $n < 30$，剪枝有效时算法运行时间在毫秒级，性能较好。

4.3 动态规划（DP）测试结果

为了系统评估 DP 算法在三分类问题上的性能，我们分别在以下三种典型场景下进行了实验：(1) 固定 n、增大 sum；(2) 固定 sum、增大 n；(3) 同时逐步提高 n 与 sum。以下对三类实验结果分别进行分析。

4.3.1 固定 n，增大 sum 的情况

在该实验中，我们固定元素数量 n，逐步增加总和 sum，并记录 DP 算法的运行时间。实验结果如图 1 所示。可以明显观察到，随着 sum 的增大，DP 的运行时间呈现非线性增长趋势，但总体与 DP 的理论复杂度 $O(n \cdot sum)$ 保持一致。

Table 2: 固定 n，增大 sum 时 DP 算法运行时间统计（秒）

$total\_sum$	mean	min	max	var
10002	0.3868	0.3765	0.4156	0.000266
20001	1.5310	1.4883	1.6145	0.003158
30000	3.3737	3.3543	3.4249	0.000853
40002	6.4288	6.3932	6.4482	0.000509
50001	11.5226	11.3661	11.6768	0.012789
60000	18.0380	16.6423	23.2785	8.589885
70002	24.0130	22.8261	26.7825	3.014150
80001	30.6717	29.9962	32.8286	1.466136
90000	37.7163	37.1789	39.1276	0.661494
100002	45.1194	43.9791	48.8319	4.401916

从数据中可以看出：

- 当 sum 较小（例如 10^4 以下）时，运行时间增长相对平缓，均值维持在 $O(0.3 \sim 0.4)$ 秒的水平。
- 随着 sum 扩展到 10^5 量级，运行时间呈明显加速增长，尤其是方差也随之增大，显示状态表规模显著增加对性能的影响。
- 对不同复杂度假设的拟合结果显示，$O(n^2)$ 拟合曲线与实验数据最为贴合（MSE $= 0.772$），拟合参数为 $[4.662 \times 10^{-9}, -0.0684]$，远优于 $O(n)$（MSE $= 9.41$）或 $O(n \log n)$（MSE $= 7.45$）的拟合效果。

实验数据及拟合曲线见图 1。

综上，固定 n 条件下，DP 算法运行时间随 sum 单调增加，趋势稳定且可预测；即便在大规模 sum 条件下，其性能仍明显优于 DFS，表现出良好的稳定性和可扩展性。

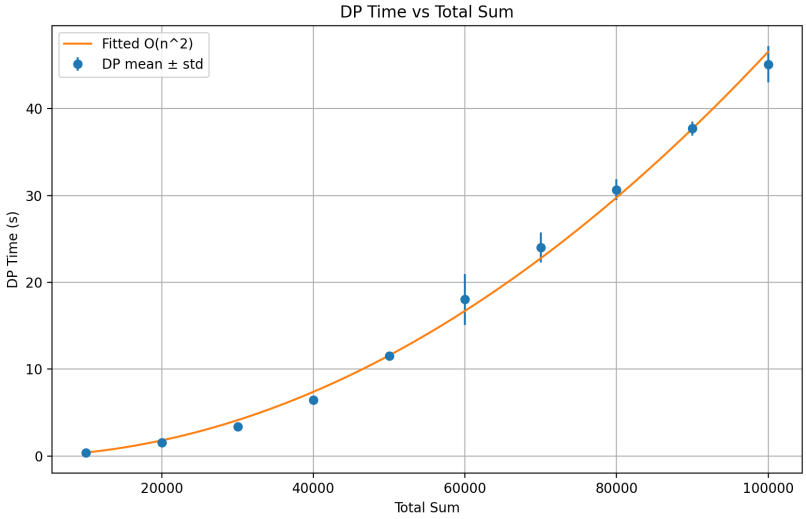

Figure 1: 固定 n，增大 sum 时 DP 算法运行时间及不同复杂度拟合曲线对比

4.3.2 固定 sum，增大 n 的情况

本组实验固定总和 sum，逐步增加输入规模 n，测试结果如图 2 所示。DP 的状态空间大小与 n 成线性关系，因此运行时间的增长主要由迭代次数增加所引起。

Table 3: 固定 sum，增大 n 时 DP 算法运行时间统计（秒）

n	mean	min	max	var
100	4.3085	4.2870	4.3247	0.000170
200	10.1568	9.9115	10.7473	0.090331
300	19.5679	15.8822	27.2218	16.318133
400	22.3316	21.8123	23.3965	0.314617
500	30.9510	28.7956	34.6313	4.063425
600	36.7893	35.6425	38.6712	1.007763
700	47.5941	43.3565	50.0603	5.547414
800	56.2361	49.5285	66.9230	38.305580
900	62.0149	58.8609	68.4820	12.228341
1000	68.6487	65.5686	76.3955	16.920649

拟合曲线 2中可以观察到：

- 在中小规模 n 范围内，DP 的运行时间基本保持线性增长，拟合结果表明 $O(n)$ 拟合较为贴合。通过最小二乘法拟合得到的时间复杂度函数为：

$$T(n) = 0.073154 \cdot n - 4.374624 \tag{4}$$

拟合的均方误差（MSE）为 12.146 仍然可以接受

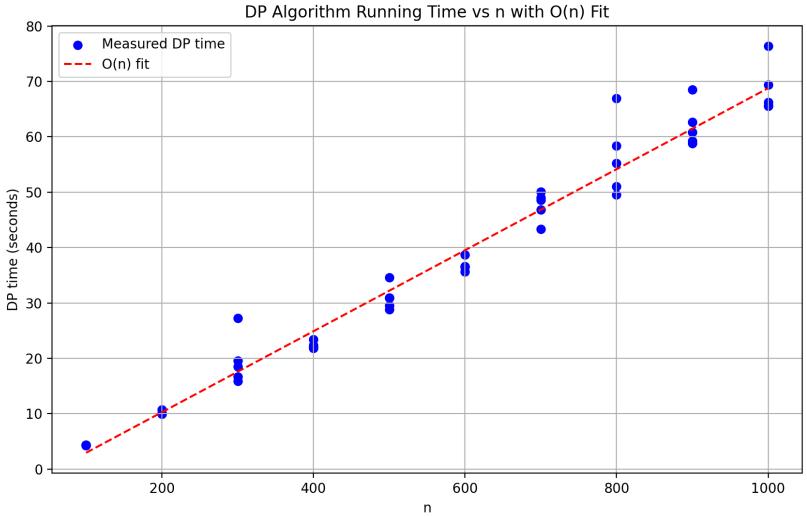

Figure 2: 固定 sum，增大 n 时 DP 算法运行时间及 O(n) 拟合曲线

4.3.3 同时增大 n 与 sum 的情况

为了进一步评估动态规划算法在大规模输入下的极限性能，我们系统地同时增大输入规模 n（项目数量）与 sum（目标和），使 DP 的状态空间呈二次方级增长（$O(n \cdot sum^2)$）。测试结果如图 3所示。

从可视化结果可以清晰观察到，当 n 与 sum 同时扩大时，DP 算法的运行时间呈快速增长趋势。具体而言：

- **指数级增长特征**：运行时间随 n 和 sum 的增大呈现明显的非线性增长，与 $O(n \cdot sum^2)$ 的理论复杂度高度一致（$R^2 = 0.9959$）。
- **硬件限制触及**：在 $n \in [50, 450]$，$sum \in [5208, 47349]$ 的测试范围内，最大运行时间已显著增长，并在部分数据点上观察到运行超时或内存溢出现象。
- **可计算性边界**：DP 算法的实际可计算范围严格受限于 $n \cdot sum^2$ 的规模，当两者同时增长时，算法很快到达不可计算区域，凸显了状态空间组合爆炸问题的本质性限制。

该实验从实证角度验证了模拟退火算法在大规模输入中具有不可替代的意义——动态规划算法虽然在小规模问题上具有最优性保证，但其计算复杂度使其难以扩展到现实世界的大规模优化问题，这强调了启发式算法在实际应用中的必要性。

4.4 模拟退火（SA）测试结果

4.4.1 base construction

首先使用 DP 算法验证一批规模较小且均可分解的基准实例，记为 $S_2, S_2, \ldots, S_m$。从中任意选取 k 个作为构造基 (可重复选择)，记为 $A_1, A_2, \ldots, A_k$，并为每个基分配不同的权位，从而构造最终的测试集

$$T = \sum_{i=1}^{k} factor_i \times A_i, \ factor_i = random(1, 7^6)$$

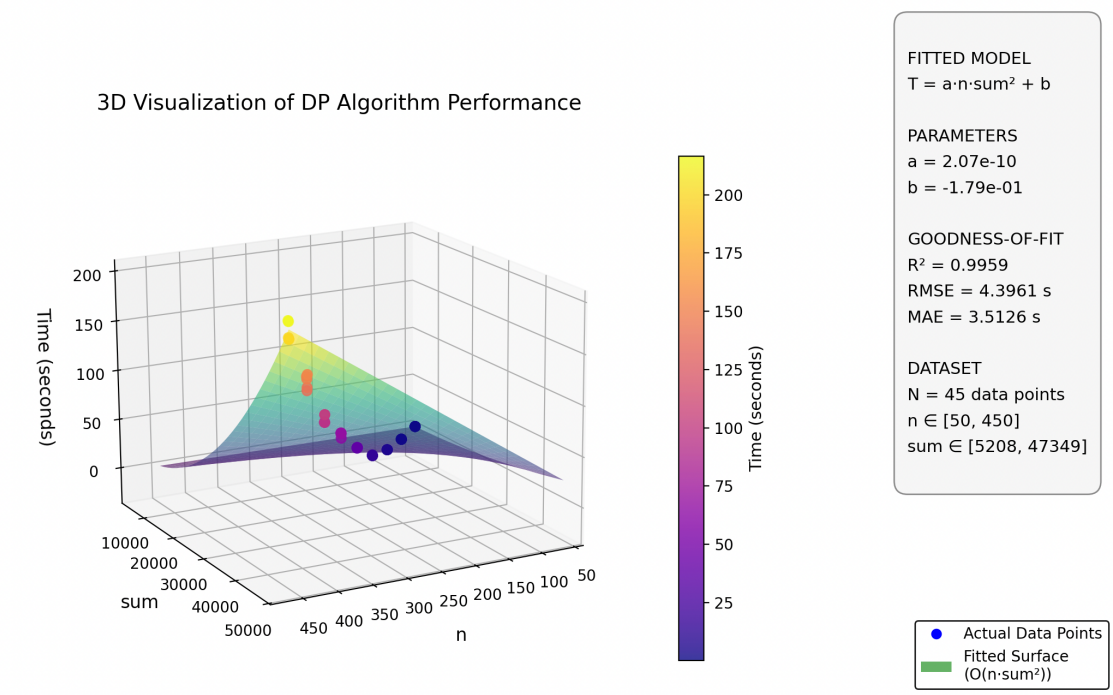

Figure 3: 动态规划算法在 n 与 sum 同时增大时的性能可视化。图中蓝色散点为实际运行时间，绿色曲面为基于 $O(n \cdot sum^2)$ 复杂度模型的拟合结果。参数拟合结果为 $T = 2.07 \times 10^{-10} \cdot n \cdot sum^2 - 0.179$，拟合优度 $R^2 = 0.9959$，表明模型能够准确描述算法的时间复杂度。

其中“×”表示将集合 A_i 中的每个元素统一乘以权重 $factor_i$。这种分层结构保证了每个 A_i 的可解性在组合后仍然得以保留.

对于每个生成的实例，脚本将数组写入 `input.txt`，随后最多运行 $K_{\max} = 10$ 次 `./sa`，直到其找到正确解为止。每次求解结束后使用 `./check` 进行正确性验证，并记录首次成功的重启次数。

在实验中, 我们取用 20 组基 (基中最大元素小于 300), 每次至少选 10 组来构造 T; 当我们测试了 10000 组解 (这样的生成方式很快) 后发现, 若最多允许 `./sa` 重启 10 次, 我们得到如下结果:

- **Total instances**: 10,000
- **Failed instances**: 73
- **Average first restart count (successful cases)**: 1.10

4.4.2 greedy fill-in

为了进一步评估模拟退火算法（`./sa`）的性能与稳定性，我们额外构建了一类 **保证一定存在可行解**的测试集。生成脚本通过严格控制总和与元素构造方式，使得每个输入实例都满足三分类问题的可解性条件。

在生成阶段，脚本固定数组长度（本实验中为 $n = 1000$），并将所有元素限制在区间 $[1, 10^6]$ 内。首先为三个桶分配平衡的目标和，随后逐步生成随机数填充各桶，同时确保不会超出目标和。生成完成后，脚本通过最后一步一致性修正，使得数组总和严格可被桶数整除，从而保证实例一定存在一个 **完美三分**方案。

对于每个生成的实例，脚本将数组写入 `input.txt`，随后最多运行 $K_{\max} = 10$ 次 `./sa`，直到其找到正确解为止。每次求解结束后使用 `./check` 进行正确性验证，并记录首次成功的重启次数。

我们得到如下的结果

4.4.3 Comparison

显然 base construction 的构造方式相对 greedy fill-in 而言其实少了很多随机性, 但是 `./sa` 竟然对 base construction 得到的解有这么好的效果, 这可能提供了一种对三分类问题增加限制条件, 使得该 `./sa` 能较好解决的思路; 此外, 尽管两者的成功率差距很大, 但是在最大重启次数为 10 时, 其平均首次成功次数竟然都非常接近 1 这说明简单的增加一定规模的重启次数不能有效的增加 SA 的性能

4.5 综合分析

- DFS+ 剪枝在小规模数据上效果显著，但在剪枝失效情况下，搜索空间指数增长，性能极差。
- DP 算法在中小规模数据上稳定高效，但受限于 n 和 sum 的乘积，超大数据可能无返回。
- SA 适用于大规模或近似求解场景，可快速得到可行解，但正确率低于 DP。

总体而言，不同算法适合不同规模和场景的三分类问题：DFS 更适合小规模精确求解，DP 适合中规模最优求解，SA 可用于大规模近似求解。

5 Bonus: 推广到 K-Partition

我们探讨了当划分份数从 3 份推广到 K 份（$K = 4, 5, \dots$）时，即 **K-Partition Problem**，上述算法的适用性及复杂度变化。

5.1 深度优先搜索 (DFS): 指数爆炸

- **适用性**: DFS 算法逻辑有通用性，只需将目标桶数从 3 改为 K 即可直接运行。
- **复杂度分析**: 每个物品现在有 K 种选择。

$$T(N, K) = O(K^N)$$

当 K 增大时，搜索树的宽度（分支因子）迅速增加，计算量呈指数级爆炸。

- **剪枝效果**: 虽然基本复杂度变差，但等效性剪枝 *(Symmetry Breaking)* 的效果会随着 K 的增加而变得更加显著。因为初始时 K 个空桶是完全等价的，这使得搜索空间减少了 $K!$ 倍（同构解的数量）。尽管如此，对于较大的 N 和 K，DFS 仍难以在有限时间内求解。

5.2 动态规划 (DP): 内存限制

- **适用性**: 理论上可行，但实际上因内存限制而不可行。
- **状态定义变化**: 为了确定前 $K-1$ 个桶的状态（最后一个桶的状态由总和约束隐含），我们需要维护 $K-1$ 个维度:

$$dp[s_1][s_2]\dots[s_{K-1}]$$

- **复杂度分析**:

$$\text{时间/空间复杂度} = O(N \cdot \text{Target}^{K-1})$$

- **实例分析**: 假设 Target $= 100$，每个维度大小为 100。
  - 当 $K = 3$ 时，空间为 $100^2 = 10,000$，内存可控。
  - 当 $K = 4$ 时，空间为 $100^3 = 1,000,000$，尚可接受。
  - 当 $K = 10$ 时，空间为 100^9，这不仅远超计算机内存物理极限，甚至超过了硬盘存储能力。

  结论: DP 算法不具备向高维推广的能力。

5.3 模拟退火 (Simulated Annealing)

- **适用性**: 模拟退火在处理 K-Partition 问题时表现出极佳的鲁棒性。只需修改目标函数和桶的个数参数，无需改变算法核心逻辑。
- **复杂度分析**:

$$T(N, K) \approx O(\text{Iterations} \cdot K)$$

  其复杂度仅随 K 线性增长（主要消耗在能量函数的计算上，若采用增量更新，甚至可以做到 $O(1)$ 与 K 无关）。
- **优势**: 它避开了 DFS 的指数级深度和 DP 的指数级空间，是解决大规模 K-Partition 问题（如 $K = 100, N = 10000$）在工程上可行的方法。

6 结论与讨论 (Conclusion)

在本项目中，我们深入研究了 3-Partition 问题，并对比了三种不同性质的算法：

1. **DFS** 在强力剪枝的辅助下，是解决中小规模（$N \leq 60$）及大数值问题的最佳精确解法。
2. **DP** 揭示了该问题的伪多项式性质，适用于 N 较大但数值总和较小的特定场景，但在通用性上受限于内存瓶颈。
3. **模拟退火**作为启发式算法，展示了在处理大规模输入（$N = 1000$）和高维变体（Bonus K-Partition）时的强大能力，虽然牺牲了完备性，但换取了极高的工程实用性。