组合最优化算法 阅读报告

郑滕飞 2401110060

目录

1 准备工作

1.1 论文选取

由于上课已经介绍了背包问题的基本知识，这个问题对应的模型相对简单，但应用很多，我决定以此问题进行论文的选取。查询可发现 2024 年的 STOC 接收了四篇关于背包问题的论文[1-4]，不过，只有第一篇是精确算法，另外三篇为近似算法。由于精确算法的分析过程包含了很多对近似求解具有启发性的思路，我们即以第一篇论文**近二次时间的 0-1 背包算法**作为主要阅读对象 (准确来说为其完整版[5])。如无特殊说明，下文的 “背包问题” 均指 0-1 背包问题，“原论文” 指这篇完整版论文。

在介绍论文的具体成果与思路前，值得一提的是，为什么在 2024 年，还有如此多对于背包问题的研究。总体来说，对于背包问题的研究有三个重要的意义：

- 从问题结构本身的角度，它关注的核心是物品权重集合，而这是一个**整数多重集合** (允许元素重复出现的集合)，在研究算法时出现的一系列数据结构与操作往往可以推广到一般多重集合的操作上，从而在存在类似数据结构的其他问题中使用。
- 从复杂度理论的角度，背包问题是一个 **NPC 问题**，而在诸多等价的 NPC 问题中它具有非常简单的描述，因此对其算法的改进利用等价性也可以转化为对其他 NPC 问题的算法改进。另一方面，如果考虑其伪多项式算法 (就像原论文中研究的)，它还可以等价为一些 P 问题，带来更多的应用。
- 从组合优化的角度，对此问题的精细处理事实上应用了极多的优化技巧，例如**从粗糙到精细**的分析手段、**自顶向下分治**的思路，或是时空之间、不同步骤时间复杂度之间的**权衡**。也正是因此，原论文研究的虽然是精确算法，但在分治的任何一个层级，都可以开发出对应的近似算法，

正因如此，虽然其问题描述非常简单，低复杂度的算法构建仍然是十分复杂的。篇幅所限，本笔记将主要关心数学处理的部分，忽略数据结构层面的实现与操作，只给出复杂度的结论。此外，本笔记中如 [引理 2.1] 的标记代表结论在**原论文中的序号**。

笔记中，虽然思路与大部分证明参考了原论文，小部分证明为独立完成，**思路性的内容与观察**则几乎全部独立完成，且对原论文结构做了不少调整。

1.2 问题与记号

首先，如无特殊说明，我们以小写字母与大写字母 C, L 表示数或向量，$\mathbb{F}^I$ 表示元素在 $\mathbb{F}$ 中、下标在指标集 I 中的向量集合；粗体小写字母 $\mathbf{a}, \mathbf{x}$ 表示矩阵或二维向量；一般的大写字母表示集合，并用绝对值符号表示其元素个数，$\sqcup$ 表**无交并** (这将在划分时大量用到)，2^A 表示 A 的子集族；手写体字母 $\mathcal{Q}, \mathcal{W}$ 等表示映射 ($\mathcal{O}$ 表示复杂度至多为此量级)，尖角体字母 $\mathfrak{X}, \mathfrak{W}$ 等表示多重集合，双线体 $\mathbb{N}$ 表自然数、$\mathbb{N}^+$ 表正整数、$\mathbb{Z}$ 表整数、$\underline{\mathbb{Z}}$ 表**整数或负无穷**。此外，所有的 log 都表示 **2 为底的对数** (虽然大部分情况下底数并不会影响估算过程)。

记从 a 到 b 的所有整数 (含两端) 集合为 $[a..b]$，并记 $[n] = [1..n]$。一个**背包问题**是指，给定总权 $t \in \mathbb{N}^+$、物品数量 $n \in \mathbb{N}^+$ 与 n 个物品的权重 $w_i \in \mathbb{N}^+$、价值 $p_i \in \mathbb{N}^+$ (指标 $i \in [n]$)，求指标集合 $X \subset [n]$ 使得约束条件

$$\mathcal{W}(X) = \sum_{i \in X} w_i \leq t$$

成立，且总价值

$$\mathcal{P}(X)=\sum_{i\in X}p_i$$

最大。

进一步地，记所有权重的最大值为 $w_{\max}$，我们只考虑**非退化的背包问题**，即满足

$$w_{\max}\le t,\quad \mathcal{W}([n])>t,\quad w_{\max}\le n^2$$

的背包问题。若第一式不成立，可以直接排除权重大于 t 而根本不可能放入背包的物品；若第二式不成立，可直接将所有物品放入背包。由于这两个操作都可以在 $\mathcal{O}(n)$ 时间完成，排除这两种情况是合理的。

对于第三个式子，由于直接进行动态规划的复杂度为 $\mathcal{O}(nt)$，若第三个式子不成立，利用 $t\le nw_{\max}$，可知 $\mathcal{O}(nt)\le\mathcal{O}(w_{\max}^2)$，稍后我们将看到，这会低于原论文实现算法的复杂度，因此直接进行动态规划即可。

记所有权重构成的集合 (不计重数) 为 W，构成的多重集合为 $\mathfrak{W}$，以 $I_{W'}$ 表示权重在 W' 中的 $i\in[n]$ 构成的指标集，$\mathfrak{W}(X)$ 表示指标集 X 中的权重构成的多重集合。

为了之后的说明方便，我们还需要定义一些多重集合相关的记号：一个**有限整数多重集合** $\mathfrak{X}$ 是指所有元素均为整数，且元素个数有限的多重集合。具体来说，将元素 x 的出现次数记为 $\mu_{\mathfrak{X}}(x)$，则其元素个数

$$|\mathfrak{X}|=\sum_{x\in\mathbb{Z}}\mu_{\mathfrak{X}}(x)$$

支集 (若 $\mathfrak{X}$ 的支集包含于 I，则称其支在 I 上)

$$\mathrm{supp}\mathfrak{X}=\{x\in\mathfrak{X}\mid\mu_{\mathfrak{X}}(x)\ge1\}$$

r 支集

$$\mathrm{supp}_r\mathfrak{X}=\{x\in\mathfrak{X}\mid\mu_{\mathfrak{X}}(x)\ge1\}$$

元素和

$$\Sigma(\mathfrak{X})=\sum_{x\in\mathbb{Z}}x\mu_{\mathfrak{X}}(x)$$

对两个多重集合 $\mathfrak{X},\mathfrak{Y}$，$\mathfrak{Y}\subset\mathfrak{X}$ 当且仅当

$$\forall x\in\mathbb{Z},\quad\mu_{\mathfrak{Y}}(x)\le\mu_{\mathfrak{X}}(x)$$

将一个多重集合的所有子集元素和记为

$$\mathcal{S}(\mathfrak{X})=\{\Sigma(\mathfrak{Y})\mid\mathfrak{Y}\subset\mathfrak{X}\}$$

而非空子集元素和为

$$\mathcal{S}^*(\mathfrak{X})=\{\Sigma(\mathfrak{Y})\mid\mathfrak{Y}\subset\mathfrak{X},|\mathfrak{Y}|>0\}$$

最后，在如上定义下，有限整数集可以视为一种特殊的有限整数多重集合，因此上述记号也可对集合使用，含义不变。此外，对一个向量定义

$$\mathrm{supp}(u)=\{i\mid u_i\ne0\}$$

1.3 论文成果

原论文的核心成果为，提出了一个复杂度为 $\mathcal{O}(n + w_{\max}^2 \log^4 w_{\max})$ 的背包问题伪多项式精确算法。

为理解此复杂度，我们定义**最大和卷积问题**为，已知向量 $a, b \in [-W..W]^{[0..n-1]}$，求

$$c[k] = \max_{i \in [0..k]} (a[i] + b[k-i]), \quad \forall k \in [0..n-1]$$

一个常用的假说是，对任何 $\varepsilon > 0$，此问题不存在 $\mathcal{O}(n^{2-\varepsilon}\text{polylog}(W))$ 复杂度的算法 (这里 polylog 代表 log 的某多项式)。在此假说下，对任何 $\varepsilon > 0$，背包问题不存在 $\mathcal{O}((n + w_{\max})^{2-\delta})$ 复杂度的算法[6]。由此，原论文的算法事实上在指数上已经达到了几乎理论最优的情况，能改进的只有 log 处的指数。

为了达到此结果，原论文事实上进行了四个层次的划分——这也是这篇阅读笔记的排版：

- **权重划分**，即从贪婪解出发，通过将权重分层，使得只需解决不同权重个数较少情况的问题，而其他情况通过优化的动态规划算法可迭代更新以较低复杂度得到；
- **位序划分**，即对上述不同权重个数较少的情况进一步以元素的位序进行划分，细化每一步的动态规划迭代，通过更高空间复杂度 (即提示集的引入) 以优化动态规划的时间，最终将每一步化为原论文提出的一个特殊优化问题：HKE 问题；
- **染色划分 I**，将一般 HKE 问题通过一定的算法进行染色划分，以控制提示集的规模较小，并利用一定的分解引理递归组合子问题的解以得到原问题的解；
- **染色划分 II**，将较小提示集的 HKE 问题通过具有分离性的染色划分为提示集大小至多 1 的 SHKE 问题，并再次利用一定的分解引理递归组合子问题的解以得到原问题的解。

最后，由于对 **SHKE 问题**可以构建复杂度较低的算法，一层层回溯即可得到原问题的一个低复杂度算法。虽然这个划分过程看似繁琐，但每一步的目的都是限制问题到更小规模以引用具有更高特殊性的算法——而停止在任何一层划分事实上都可以得到某个近似算法，这将在之后每步的分析中说明。

2 权重划分

本部分中，我们将先利用贪婪算法对问题的规模进行最基本的限制，再介绍直接动态规划与它的一个快速算法，并以此在特定的权重划分下给出迭代，控制总复杂度。

2.1 贪婪解

首先，为了进一步规范问题，我们定义物品 i 的**效用**为 w_i/p_i，则可以证明，原问题可在 $\mathcal{O}(n)$ 时间内转化为最优解相同的另一个背包问题，使得 n、$w_{\max}$ 与 t 均不变，且所有物品的价值 p_i、效用 w_i/p_i 均**互不相同** [引理 2.1]。处理的基本思路为，为将所有 p_i 放大足够大的倍数后，给结果增加 i 以保证余数不同 (从而新的 p_i 互不相同)，再进一步放大保证效用不同。

由此，我们不妨假设原问题满足上述性质，于是接下来叙述的贪婪解与位序都具有**唯一性**，可以进行确定的操作。

为了对问题进行初步的规模缩减，我们需要先考虑最常规的**贪婪算法**。假设所有物品的效用排序为

$$w_{k_1}/p_{k_1} > w_{k_2}/p_{k_2} > \cdots > w_{k_n}/p_{k_n}$$

我们记贪婪解

$$G=\{k_1,\dots,k_r\},\quad \sum_{i=1}^{r}k_i\le t,\quad \sum_{i=1}^{r+1}k_i>t$$

由于非退化假设，此贪婪解定义合理。

必须说明的是，虽然贪婪解的定义里出现了**排序**这个复杂度 $\mathcal{O}(n\log n)$ 的操作，实际上其可以通过线性时间中位数查找算法 [注 2.2] 对 G 进行构造，从而达到 $\mathcal{O}(n)$：先用线性时间计算效用序列的中位数，设时间复杂度 Cn，遍历序列计算效用高于中位数的质量求和，可在 $(C+1)n$ 时间得到效用最大的 $[n/2]$ 个物品权重和是否 t。若是，则考虑前 $[n/2]$ 个物品的子问题；若否，则考虑后 $n-[n/2]$ 个物品的权和为 $t-\sum_{i\le[n/2]}w_{k_i}$ 的子问题。以此，最终可以得到一个不超过 $4(C+1)n$ 时间复杂度的算法，且此过程已经得到了确定的 G。

此后，我们用 $\bar{G}$ 表示 $[n]\backslash G$，则问题可以转化为，从 $\bar{G}$ 找一个集合 A 放入贪婪解、从 G 中找一个集合 B 拿出贪婪解，使得如此更新后不会超重，且价值增加最多。将满足这样条件的 (A,B) 称为一个**交换对**，最优解对应的交换对称为最优交换对。

我们下面说明，这样的初步贪婪解确实是对问题**规模缩减**有意义的。具体来说，由于 G 对于承重 $\mathcal{W}(G)$ 的背包一定为最优解，对任何最优交换对 (A,B) 与任何最优解 X 有直接的估算：

$$0\le\mathcal{W}(A)-\mathcal{W}(B)=\mathcal{W}(X)-\mathcal{W}(G)<w_{\max}$$

$$S^*(\mathfrak{W}(A))\cap S^*(\mathfrak{W}(A))=\varnothing$$

这里第二式成立是由于若有子集质量和相同，同时删去应更优。利用这些直接的估算，可以得到对解的规模的估算：

$$|A|+|B|\le 2w_{\max}$$

从而还有

$$\mathcal{W}(A)+\mathcal{W}(B)\le 2w_{\max}^2$$

证明的思路是直观的，即利用组合方法，从子集不存在重复的和得到元素个数不能过多。

从贪婪解出发考虑交换对的最大意义是，将问题从 n 相关规模的转化为了 $w_{\max}$ 相关的规模，而非退化情形 $w_{\max}$ 不会过大，这就给出了**腾挪复杂度的空间**。

2.2 分割权重

很显然，贪婪算法给出的粗略解与精确解可能仍然存在很大的距离。因此，我们仍需要逐步得到精确解。就个人搜集到的背包问题的相关算法而言，**一切伪多项式精确算法本质上都是动态规划**。由于背包问题清晰的最优子问题特性，用空间换时间的思路进行求解本身已经近乎最优了。得益于 SMAWK 算法，对于权重相同的物品，动态规划复杂度可以很低 (将在下一部分介绍)，因此以权重进行划分并迭为成为了自然的想法。

为了使此操作可以进行，我们需要一个从多重集合研究中得到的引理：存在常数 C 使得，只要支在 $[m]$ 上的两多重集合 $\mathfrak{A}$、$\mathfrak{B}$ 满足

$$|\mathrm{supp}\mathfrak{A}|\ge Cm\log m,\quad \Sigma(\mathfrak{B})\ge\frac{Cm^2\sqrt{\log m}}{|\mathrm{supp}\mathfrak{A}|}$$

即有 $S^*(\mathfrak{A})\cap S^*(\mathfrak{B})\neq\varnothing$ [引理 3.1]。

由此引理可以证明，能将权重集合 W 划分为 $W_1 \sqcup W_2 \sqcup \cdots \sqcup W_s$，使得 ($C$ 与引理中一致) 对任何最优交换对 (A, B) 有

$$s < \log_2 \sqrt{w_{\max}}$$

$$\forall j \in [s], \quad |W_j| \le 4C\sqrt{w_{\max} \log w_{\max}} 2^j$$

$$\forall j \in [s], \quad \mathcal{W}(A \cap I_{W_{>j}}) \le 4C w_{\max}^{3/2} 2^{-j}$$

$$\forall j \in [s], \quad \mathcal{W}(B \cap I_{W_{>j}}) \le 4C w_{\max}^{3/2} 2^{-j}$$

这里 $W_{>j} = W_{j+1} \sqcup \cdots \sqcup W_s$，后面出现的 $W_{\le j}$ 则表示 $W_1 \sqcup \cdots \sqcup W_j$，划分所需的时间复杂度为 $\mathcal{O}(n + w_{\max} \log w_{\max})$ [引理 3.2]。

这里只介绍数学上的构造，算法上的实现可完全类似利用线性时间中位数查找进行。我们假设所有物品已经按效用从大到小排好序，贪婪解对应的 $G = [i^*]$，取 l_j 为使得

$$|\mathrm{supp}\mathfrak{W}([l_j..i^*])| \le 2C\sqrt{w_{\max} \log w_{\max}} 2^j$$

的最小 l_j，r_j 为使得

$$|\mathrm{supp}\mathfrak{W}([i^* + 1..r_j])| \le 2C\sqrt{w_{\max} \log w_{\max}} 2^j$$

的最大 r_j，并令

$$W_{\le j} = \mathrm{supp}\mathfrak{W}([l_j..r_j])$$

根据定义即可发现这种分割满足第二条性质，而利用 C 的界即可得到至多 $[\log_2 w_{\max}]$ 层即包含全部元素。进一步地，通过引理可以估算得到，若第三条不成立，则 A 与 B 将存在和相同的子集，或可以单独更新 A 的元素使得结果更优矛盾，对第四条同理可证。

注意到，如此进行分割的好处在于，随着 j 的增大，集合**包含的权重个数指数增多**，但**可选的权重指数减小**，且**总集合个数为对数量级**。这样的指数增加-减小保证了实际算法的量级控制在多项式，而且是较低次的多项式。在进行位序划分时，还将再次运用类似的结构。

2.3 动态规划

最后，我们来给出一般的动态规划表 (后称 **DP 表**，即 Dynamic Programming Table) 的定义，并且给出在 W_1 对应的解已知时，如何迭代得到后续问题的解，且总复杂度可以控制。这样，第一层划分后，我们就只需要解决 W_1 对应的子问题了。

先回到之前对于交换对的讨论。我们定义一个交换对 (A, B) **支在** I 上当且仅当 $A \cup B \subset I$，记作 $\mathrm{supp}(A, B) \subset I$，而它 I-最优则指存在最优交换对 (A^*, B^*) 使得 $A = A^* \cap I$，$B = B^* \cap I$。

有了交换对后，我们对动态规划的描述也可以通过它们进行。首先，与传统的二维数组动态规划不同，我们将直接进行的动态规划看作一个**每步更新的一维数组**。具体来说，一个长 L (虽然其实际上可能有 $2L + 1$ 个有效值) 的 DP 表 q 是指 (这里右侧 $[z]$ 等效于下标 z)

$$\{q \in \mathbb{Z}^{\mathbb{Z}} \mid \forall |z| > L, q[z] = -\infty\}$$

对任何 $I \subset [n]$，称 q 是 I-有效的当且仅当

$$\forall z \in \mathbb{Z}, q[z] \ne -\infty, \quad \exists (A', B'), \quad \mathrm{supp}(A', B') \subset I, \quad \mathcal{W}(A') - \mathcal{W}(B') = z, \quad \mathcal{P}(A') - \mathcal{P}(B') = q[z]$$

称 q 是 I-最优的当且仅当其 I-有效，且存在 $z \in \mathbb{Z}$、$q[z] \ne -\infty$ 与 I-最优的 (A', B') 使得

$$\mathcal{W}(A') - \mathcal{W}(B') = z, \quad \mathcal{P}(A') - \mathcal{P}(B') = q[z]$$

我们简单介绍一下直接的动态规划如何利用此表进行——虽然这并不会在后文用到。从 $\varnothing$-有效的长为 0 的 DP 表 $q[0]=0$ 出发，设当前已经更新了 $i-1$ 个物品，长度为 L，则第 i 次更新执行

$$\forall z\in[-L-w_i..L+w_i],\quad q[z]=\begin{cases}\max\{q[z],q[z-w_i]+p_i\} & i\in\bar{G}\\ \max\{q[z],q[z+w_i]-p_i\} & i\in G\end{cases}$$

与通常方法类似可证明这样的动态规划在第 i 步迭代后得到了一个 $[i]$-最优的长为 $\mathcal{W}([i])$ 的 DP 表，由此 n 次迭代可得最优解，复杂度为对 w_i 求和了 $n-i+1$ 次，从而为 $\mathcal{O}(n^2w_{\max})$。

有两件事必须注意：

- 此动态规划算法的复杂度相较直接动态规划貌似从 $\mathcal{O}(nt)$ 上升到了 $\mathcal{O}(n^2w_{\max})$，但剥离出 t 后，复杂度可以通过稍后介绍的办法进行极大的改进。
- 复杂度达到 $n^2w_{\max}$ 可以粗略看作所有物品的权重都近乎是 $w_{\max}$，而在这种情况下，贪婪解达到的结果会很接近最优，理应有进行简单判断的方式。

正是出于这两个观察，我们试图将**横向**的每次对于某物品更新改进为**纵向**的每次对于某权重更新。有趣的是，纵向的更新恰好可以转化为之前提到的最大和卷积问题 (但有一定的**凸性**要求，即 $b[i]-b[i-1]\geq b[i+1]-b[i]$)，而这又可以用 **SMAWK 算法**快速求解，从而得到如下引理：

设 $I\subset[n]$，$J\subset\bar{G}$ 或 $J\subset G$ 使得 $I\cap J=\varnothing$，且 $\operatorname{supp}\mathfrak{W}(J)=\{w\}$。假设已知对所有 $I\cup J$-最优交换对 (A',B') 均有 $|W(A')-W(B')|\leq L'$，则可在 $\mathcal{O}(L+L'+|J|+w_{\max}\log w_{\max})$ 的时间复杂度将一个 I-最优的长为 L 的 DP 表更新为一个 $I\cup J$-最优的长为 L' 的 DP 表 [引理 2.7]。

对这个引理的证明实际上是直接给出了对应的算法，不过，它除了需要用到我们将在 6.1 节详细介绍的 SMAWK 算法外，还需要将在 3.1 节介绍的位序相关的引理。因此，即使直观来说同权重的物品应该容易更新，对此问题复杂度的压缩实际上比直观困难得多，我们将在 6.1 节论述其大致过程，作为 SMAWK 算法的一个应用。

若用此算法对单个物品进行更新，复杂度 $\mathcal{O}(L+w_i+w_{\max}\log w_{\max})$ 显然超过了直接更新的复杂度。但是，$|J|$ 的个数越大，即**权重重复越多**时，此算法的效率就会相对越高。将此算法应用在分解后的权重其实是一个相对直接的过程——我们先假设 I_{W_1}-最优的 DP 表 q 已知，然后对每个 $j\in[2..s]$，从 $I_{W_{j-1}}$-最优的 DP 表更新至 I_{W_j}-最优的 DP 表。

具体的更新方式为，设 $L_j=4Cw_{\max}^{3/2}2^{-j}+w_{\max}$，则通过定义可以证明，对 $j\in[2..s]$，任何最优交换对 (A,B) 有

$$|\mathcal{W}(A\cap I_{W_{\leq j}})-\mathcal{W}(B\cap I_{W_{\leq j}})|\leq|W(A)-W(B)|+|\mathcal{W}(A\cap I_{W_{>j}})|+|\mathcal{W}(B\cap I_{W_{>j}})|\leq L_j$$

且再拆分可发现 (注意下式对 $j=1$ **并不成立**，这也是为什么第一步更新无法从空集出发通过此迭代算法得到)

$$|\mathcal{W}(A\cap I_{W_{\leq j}})-\mathcal{W}(B\cap I_{W_{\leq j-1}})|\leq L_{j-1}$$

由此，我们可以先将一个 $I_{W_{\leq j-1}}$-最优的 DP 表收缩到长 L_{j-1} (将其范围外的点置为 $-\infty$，从算法上来说忽略即可，无需额外复杂度)，不会影响最优性。

接下来，我们记 $I_{W_j}^+=I_{W_j}\cap\bar{G}$、$I_{W_j}^-=I_{W_j}\cap G$，先对 $I_{W_j}^+=I_{W_j}\cap\bar{G}$ 中的每个权重进行更新，再对 $I_{W_j}^-=I_{W_j}\cap G$ 中的每个权重进行更新。由于每步都保持了 L_{j-1} 的长度上界，根据上方引理可知总复杂度为

$$\sum_{w\in W_j}\mathcal{O}(L_{j-1}+|I_{\{w\}}\cap\bar{G}|+w_{\max}\log w_{\max})+\sum_{w\in W_j}\mathcal{O}(L_{j-1}+|I_{\{w\}}\cap G|+w_{\max}\log w_{\max})$$

也即

$$\mathcal{O}(|W_j|(L_{j-1}+w_{\max}\log w_{\max})+|I_{W_j}|)$$

对每一层如此更新也即总复杂度

$$\sum_{j=2}^{s}\mathcal{O}(|W_j|(L_{j-1}+w_{\max}\log w_{\max})+|I_{W_j}|)$$

第二项求和即为 n，将第一项与 s 直接代入权重划分的结论可知总复杂度为 $\mathcal{O}(w_{\max}^2\log^{3/2}w_{\max}+n)$。

3 位序划分

在上一部分，我们已经得到，若 I_{W_1}-最优的 DP 表已知，则利用动态规划完成整个算法需要

$$\mathcal{O}(w_{\max}^2\log^{3/2}w_{\max}+n)$$

的复杂度。

本部分我们将初步讨论 I_{W_1}-最优的 DP 表如何得到。利用位序划分，I_{W_1}-最优 DP 表的求解可以转化为若干个 HKE 问题，而若假定解法已知，我们事实上可以得到 I_{W_1}-最优 DP 表的求解复杂度为

$$\mathcal{O}(w_{\max}^2\log^4 w_{\max}+n)$$

综合以上两步即可以得到最终的复杂度结论。

3.1 位序

自然地，想求解 I_{W_1}-最优 DP 表，我们需要先观察 W_1 的性质。根据之前划分时所证明的，它的性质其实只有一个，也即

$$|W_1|=\mathcal{O}(w_{\max}\log w_{\max})$$

对比 2.1 节处理后，我们发现，这的确又是一次**问题规模的缩减**：从数量不确定的不同权重数目变成了数量可以控制的不同权重数目。非常自然地，我们希望对**用同一权重的物品更新 DP 表**提出更好的方法，这样就有希望更快解决此问题了。

为了对比同一权重的物品，我们需要引入**位序**的概念。物品 i 的位序定义为 (回顾之前，我们已经假设了所有 p_i 互不相同)

$$\operatorname{rank} i=\begin{cases}|\{j\in G\mid p_j<p_i,w_j=w_i\}|+1 & i\in G\\ |\{j\in \bar{G}\mid p_j>p_i,w_j=w_i\}|+1 & i\in \bar{G}\end{cases}$$

用更通俗的话说，若 i 在 G 中，$\operatorname{rank} i$ 是 G 同权重的物品按价值从小到大排列后 i 所在的位置，而若 i 在 $\bar{G}$ 中，$\operatorname{rank} i$ 是 $\bar{G}$ 中同权重的物品按价值从大到小排列后 i 所在的位置。由定义可直接看出，位序越低的 G 中物品应该越优先拿出，而位序越低的 $\bar{G}$ 中物品应该越优先放入，也即，对最优交换对 (A,B)，若 $i\in A$，则

$$\{i'\in\bar{G}\cap I_{w_i}\mid \operatorname{rank} i'\le\operatorname{rank} i\}\subset A$$

若 $i\in B$，则

$$\{i'\in G\cap I_{w_i}\mid \operatorname{rank} i'\le\operatorname{rank} i\}\subset B$$

由于 A 与 B 的元素个数和限制，我们还可以知道，对 $i \in A \cup B$ 有

$$\operatorname{rank} i \le 2w_{\max}$$

类比权重划分的过程与结论，对位序划分我们也需要多重集合相关的引理：存在常数 C 使得，只要支在 $[m]$ 上的两多重集合 $\mathfrak{A}$、$\mathfrak{B}$ 满足对某个 $r \ge 1$ 有

$$|\operatorname{supp}_r \mathfrak{A}| \ge C\sqrt{\frac{m\log(2m)}{r}}$$

$$\Sigma(\mathfrak{B}) \ge \Sigma(\mathfrak{A}) - m$$

即有 $S^*(\mathfrak{A}) \cap S^*(\mathfrak{B}) \ne \varnothing$ [引理 3.6]。

从引理的条件与结论能看出，它用于估计的方式事实上和 2.2 节的引理完全类似。我们先对位序作出**显式**的划分：定义 $k = \lceil \log(2w_{\max}+1) \rceil$，$J_0 = \varnothing$，且

$$\forall j \in [k], \quad J_j = \{i \in I_{W_1} \mid 2^{j-1} \le \operatorname{rank} i \le 2^j - 1\}$$

并记

$$J_{\le j} = J_1 \sqcup J_2 \sqcup \cdots \sqcup J_j$$

用上标 $+$ 表示一个集合与 $\bar{G}$ 的交，上标 $-$ 表示一个集合与 G 的交。

上述划分有结论，存在常数 C 使得对任何 I_{W_1}-最优的交换对 (A', B') 满足

$$A' \subset J^+_{\le k}, \quad B' \subset J^-_{\le k}$$

$$\forall j \in [k], \quad |A' \cap J^+_{\le j}| < m_j, \quad |B' \cap J^-_{\le j}| < m_j$$

$$\forall j \in [k], \quad |\{w \in W_1 \mid I_{\{w\}} \cap J^+_{j-1} \subset A, I_{\{w\}} \cap J^+_j \ne \varnothing\}| \le b_j$$

$$\forall j \in [k], \quad |\{w \in W_1 \mid I_{\{w\}} \cap J^+_{j-1} \subset A, I_{\{w\}} \cap J^+_j \ne \varnothing\}| \le b_j$$

$$\forall j \in [0..k], \quad m_j = C2^{j/2}\sqrt{w_{\max}\log(2w_{\max})}, \quad b_j = C2^{-j/2}\sqrt{w_{\max}\log(2w_{\max})}$$

第一条性质利用 $\operatorname{rank} i \le 2w_{\max}$ 的估计可直接得到，第二条性质的证明须利用引理推矛盾，而第三条性质则来自位序的性质与 W_1 个数的上界 (这意味着在原规模直接进行位序划分是无意义的，也是前一步划分**无法简化**)。注意到，这里的划分也符合了**容量指数增长、有效量指数减少、总数为对数量级**三个性质，因此可以期待，只要解决了每部分对应的子问题，迭代累加后仍可保证复杂度可控。

3.2 提示集

然而，如果对每一部分仍然用之前的动态规划方法，事实上完全没有用到位序的性质，从而不能降低复杂度。要开发合适的算法，需要研究清楚位序到底带来了什么新的性质，而这其实是简单的：例如，只要 J_1 中对某个权重 w 有元素没有选中，那 J_2 中此权重对应的元素就**不可能**选中。用更数学的语言来描述，位序起到的作用其实是**排除候选**，这就启示我们，通过某种方式**将候选信息也加入 DP 表**，或许能得到对位序划分的更好处理方法。

由此，我们定义一个带提示集的动态规划表 (后称 **HDP 表**，H 指 Hinted) 为，在 DP 表 $q[z]$ 的基础上添加

$$V_q = \{z \mid q[z] \ne -\infty\}, \quad S^+ \in (2^{W_1})^{V_q}, \quad S^- \in (2^{W_1})^{V_q}$$

也即给 $q[z]$ 每个有效位置附加两个 W_1 的子集 (若 $q[z]=-\infty$，我们定义 $S^{\pm}[z]=\varnothing$ 以方便后续分析)。

HDP 表的**长度**即 q 的长度，将所有 $|S^{+}[z]|$ 的一个上界称为**正提示规模**，而 $|S^{-}[z]|$ 的一个上界称为**负提示规模**。定义一个 HDP 表 J-最优 (由于我们只考虑 W_1 的情况，设 $J\subset W_1$) 当且仅当 q 是 J-有效的，且存在 $z\in\mathbb{Z}$ 使得：

- 存在 J-最优的 (A',B') 满足

$$\mathcal{W}(A')-\mathcal{W}(B')=z,\quad \mathcal{P}(A')-\mathcal{P}(B')=q[z]$$

- 任意 I_{W_1}-最优的 (A',B')，若

$$\mathcal{W}(A'\cap J)-\mathcal{W}(B'\cap J)=z$$

  则

$$A'\backslash J\subset I_{S^{+}[z]},\quad B'\backslash J\subset I_{S^{-}[z]}$$

这里第一条性质保证了 q 是 J-最优的，而第二条性质则是对 $S^{\pm}[z]$ 作为**扩充候选**的清晰刻画。

可以想到，如果对 HDP 表能进行较好的更新，也理应存在计算 I_{W_1}-最优 DP 表的快速方法。事实上，沿用 3.2 节记号，并记 $L_j=m_jw_{\max}$，更新 HDP 表的复杂度结论为：

- 给定长为 L_{j-1} 的 $J_{\le j-1}$-最优 HDP 表，其正负提示规模均为 b_j，则可以在 $\mathcal{O}(L_jb_j\log^2(L_jb_j)+|J_j^{+}|)$ 的时间复杂度计算出一个长为 L_j、正提示规模 b_{j+1}、负提示规模 b_j 的 $J^{+}_{\le j}\cup J^{-}_{\le j-1}$-最优 HDP 表；

- 给定长为 L_j 的 $J^{+}_{\le j}\cup J^{-}_{\le j-1}$-最优 HDP 表，其正提示规模为 b_{j+1}、负提示规模为 b_j，则可以在 $\mathcal{O}(L_jb_j\log^2(L_jb_j)+|J_j^{-}|)$ 的时间复杂度计算出一个长为 L_j、正负提示规模 b_{j+1} 的 $J_{\le j}$-最优 HDP 表。

对任何 $j\in[k]$ 成立 [引理 3.9]。

这两步更新中的长度限制可以直接由位序划分的第二条性质得到，而初始长度为 0，因此 $L_0\ge 0$ 也可以成立，从而上述方法**可以从空集开始** (这与权重划分的结论并不相同)。

利用位序划分的第一条性质，上面两步重复 k 次即得到了 I_{W_1}-最优的 DP 表，而复杂度为

$$\sum_{j=1}^{k}\left(\mathcal{O}(L_jb_j\log^2(L_jb_j)+|J_j^{+}|)+\mathcal{O}(L_jb_j\log^2(L_jb_j)+|J_j^{-}|)\right)$$

利用 L_j、b_j 的定义与位序划分的性质直接估算求和可以得到最终复杂度为

$$\mathcal{O}(w_{\max}^2\log^4 w_{\max}+n)$$

至此，我们只要证明了更新 HDP 表的复杂度结论，先更新出 I_{W_1}-最优的 DP 表，再一步步扩充，即可达到 $\mathcal{O}(w_{\max}^2\log^4 w_{\max}+n)$ 的总复杂度。

不过，虽然这个候选与扩充候选的思路较为直观，要证明其更新存在快速算法并不简单，因为访问与更新候选本身也需要时间复杂度，从而仍然需要精确的**权衡**。为解决此问题，原论文将其转化为了等价问题，即下面要介绍的 HKE 问题 (原论文中的 HintedKnapsackExtend^{+})。

3.3 HKE 问题

我们先介绍**长度**为 L、**规模**为 b 的 HKE 问题的定义：设 $U \subset W_1$，对任何 $w \in U$，给定一个凹函数 $\mathcal{Q}_w : \mathbb{N} \to \mathbb{Z}$，并假设其值可以在常数时间计算。给定长为 L 的 DP 表 q，并给每个 $z \in [-L..L]$ 定义一个 U 的子集 $S[z]$，满足 $|S[z]| \le b$，且 $q[z] = -\infty$ 时 $S[z] = \varnothing$。考虑每个 z 处定义一个向量 $\mathbf{x}[z] \in \mathbb{N}^U$ 所形成的双重向量 $\mathbf{x}$，其元素记为 $\mathbf{x}[z, w]$。并定义向量 z、r 满足

$$\forall i \in [-L..L], \quad z_x[i] = i - \sum_{w \in U} w\mathbf{x}[i, w]$$

$$\forall i \in [-L..L], \quad r_x[i] = q[z_x[i]] + \sum_{w \in U} \mathcal{Q}_w(\mathbf{x}[i, w])$$

将使得 $r_x[i]$ 达到最大的 $\mathbf{x}[i]$ 集合记为 X_i，要求输出双重向量 $\mathbf{x}$，使得若

$$\forall \mathbf{x}[i] \in X_i, \quad \operatorname{supp}(\mathbf{x}[i]) = \{w \mid \mathbf{x}[i, w] \ne 0\} \subset S[z_x[i]]$$

则输出的 $\mathbf{x}[i]$ 需要属于 X_i，否则可以任意输出 $\mathbf{x}[i]$。

若去掉任意输出的部分，此问题事实上与直接更新 DP 表在某种意义上等价，类似 2.3 节中我们尚未证明的引理即可说明其复杂度为 $\mathcal{O}(|U|L)$。不过，在有了控制规模的**松弛**条件后，我们可以假设所有输出的 $\mathbf{x}[i]$ 均满足 $\operatorname{supp}(\mathbf{x}[i]) \subset S[z_x[i]]$，从而事实上可以假定输出规模在 $\mathcal{O}(bL)$ [注 3.6]，这就有将复杂度降至与 $|U|$ 无关的可能。

对此问题的求解需要再进行两次划分，本节我们先给出结论：长度为 L、规模为 b 的 HKE 问题存在复杂度为

$$\mathcal{O}(Lb \log^2(Lb))$$

的求解算法 [定理 3.11]。

接下来，我们解释如何用此算法进行 3.2 节中 HDP 表的更新。这里详细介绍第一种情况，即给定长为 L_{j-1} 的 $J_{\le j-1}$-最优 HDP 表，其正负提示规模均为 b_j 时将 J_j^+ 更新入 HDP 表，第二种情况类似操作即可。

对任何 w，设

$$\mathcal{Q}_w(x) = \begin{cases} \sum_{t=1}^{x} p_{i_t} & x \in [0..m] \\ \mathcal{Q}_w(m) & x > m \end{cases}$$

这里

$$J_j^+ \cap I_{\{w\}} = \{i_1, \ldots, i_m\}, \quad p_{i_1} > \cdots > p_{i_m}$$

由定义可发现此的确为凹函数，且花费 $\mathcal{O}(|J_j^+| + |W_1| w_{\max} \log w_{\max})$ 遍历 J_j^+、排序并累加后，可以以常数时间计算所有 $\mathcal{Q}_w$。具体来说，先遍历一次 J_j^+ 并按 w 分组，注意由 k 的界 J_j 中元素个数不会超过 $\mathcal{O}(w_{\max})$，因此对每组的排序都在 $\mathcal{O}(w_{\max} \log w_{\max})$，累加在 $\mathcal{O}(w_{\max})$，而组数不超过 W_1，即得到总和。

将 S^+ 作为 S、q 作为 q，求解长为 L_j (此时 q 的长为 L_{j-1}，但由定义也可以看作长为 L_j 的 DP 表)、规模为 b_j 的 HKE 问题，则以 $\mathcal{O}(L_j b_j \log^2(L_j b_j))$ 时间复杂度得到了解 $\mathbf{x}$ 与对应的 r、z。

此时，r 已经成为了 $J_{\le j}^+ \cup J_{\le j-1}^-$-最优 DP 表，更新它对应的提示集 $T^{\pm}$ 为，对任何 $i \in [-L_j..L_j]$，若 $r[i] = -\infty$ 则忽略，否则更新

$$T^-[i] = S^-[z[i]], \quad T^+[i] = \{w \in W_1 \mid |I_{\{w\}} \cap J_j^+| = \mathbf{x}[i, w], |I_{\{w\}} \cap J_{j+1}^+| = 0\}$$

最后，若 $|T^+[i]| > b_{j+1}$，直接将 $r[i]$ 置为 $-\infty$ 且舍弃对应的 $T^{\pm}[i]$。

上述过程中事实上能看出 HKE 问题里 $\mathbf{x}$、z、r 的含义：$\mathbf{x}$ 作为最优参数，在等价的 HDP 问题中代表某种特定要求物品的**个数**，z 对应直接的动态规划中新下标处的结果对旧下标处的**依赖**，而 r 则对应直接 DP 表更新后的**最优值**——这也体现了为什么不包含松弛条件的 HKE 问题等价于动态规划。更新提示部分的时间复杂度可以达到 (注意访问所有 $\mathbf{x}[i,w]$ 事实上空间复杂度 $\mathcal{O}(b_jL_j)$，而其他至多需要遍历一遍 J_j^+) $\mathcal{O}(|J_j^+| + |b_jL_j|)$，由此预处理、HKE 算法、更新三部分复杂度之和为

$$\mathcal{O}(L_jb_j\log^2(L_jb_j)) + \mathcal{O}(|J_j^+| + |W_1|w_{\max}\log w_{\max}) + \mathcal{O}(|J_j^+| + |b_jL_j|)$$

再利用 $|W_1|$、L_j、b_j 的界即得复杂度要求满足。

此算法正确性需要一些细致的讨论，尤其是最后更新提示集的部分，但总体都是由位序的**排除候选**性质得到矛盾。

4 染色划分 I

至此，我们将背包问题通过两次规模缩减，转化为了 HKE 问题。在 $b = |U|$ 时，HKE 问题即等价转化为了普通的动态规划，因此，可以想象其有效算法更倾向于在 b 较小时出现。于是，既然原本的 HKE 问题并不容易求解，我们就有了进一步划分的动机：将其划分为较小的 b 构成的 HKE 问题。

本部分，我们默认 HKE 问题存在 (将在下一部分证明)

$$\mathcal{O}(Lb^2(b + \log L))$$

的算法 [引理 4.8]，并以此构造 $\mathcal{O}(Lb\log^2(Lb))$ 的算法。可以发现，前者只有在 b 非常小时，复杂度才会低于后者 (一个简单的观察是 $b = \mathcal{O}(\log L)$ 时满足)，这也是为什么应用前者需要将 b 划分至很小。

4.1 染色引理

本部分与下一部分的分解与组合，相比前两部分会采取另一种思路。如果说前两部分的划分是一种**自顶向下**的分治，本部分与下一部分的划分则都是某种**平行**的分割。这是因为，只有在平行的情况下，才能保证所有的 b 都可以被控制规模。

将一个集合作某些平行的划分，事实上对应一种**染色**的方案，由此，有下面从染色研究中得到的引理：给定正整数 r 与 m 个集合 $S_1, \ldots, S_m$ 满足

$$\forall i \in [m], \quad S_i \subset [n], \quad |S_i| \le r\log(2m)$$

则存在映射 (称为 r **染色**) $\mathcal{C} : [n] \to [r]$ 使得

$$\forall i \in [m], \quad \forall c \in [r], \quad |\{j \in S_i \mid \mathcal{C}(j) = c\}| \le \mathcal{O}(\log m)$$

且构造时间复杂度为 $\mathcal{O}(mr\log m\log r + n\log r)$ [引理 4.9]。

此染色方案的证明需要用到集合**平衡**定理，即对上述的 $S_1, \ldots, S_m$，记上界为 $b = r\log(2m)$，存在 $\mathcal{O}(n + bm)$ 时间复杂度的算法确定 $x_1, \ldots, x_n$ 使得

$$\forall i \in [n], \quad x_i \in \{-1, 1\}$$

$$\forall i \in [m], \quad \left|\sum_{j \in S_i} x_j\right| \le 2\sqrt{b\log(2m)}$$

原定理中界为 $\ln(2m)$，替换为 $\log(2m)$ 后会增大，从而仍成立 [定理 B.4]。

此算法事实上意味着可以构造一种要求的 2 染色，由此，将染色后的指标集合记为 I^a, I^b，每个 S_i 都可以剖分成 S_i^a、S_i^b，考虑 a, b 对应的子问题并分别进行 2 染色即得到了一种符合要求的 2^k 染色。进一步取 $k = [\log r]$，则 2^k 染色 (可假设 r 中剩下的颜色不在像集中) 已经符合要求。这里算法正确性与复杂度都是可以直接估算得到的。

从形式上能直接看出，此染色方案将较大的集合 S_i 划分为了 r 个规模为 $\mathcal{O}(\log m)$ 的集合，于是可以应用较小的 b 的情况进行求解。

4.2 部分更新

从染色出发，我们可以较快得到问题的某种**划分**。但是，只有划分后能正确**组合**为最终结果，我们才能保证算法的复杂度。

先引入一些形式化的记号。将一个 HKE 问题记作 $K = (U, \mathcal{Q}_w, S, q)$，这里 U 为 w 的集合，$\mathcal{Q}_w$ 为所有给定的凹函数，其中 $w \in U$，S、q 即为每个位置 z 处给定集合与数。当然，我们仍然可以谈论它的长度与规模。将一个 HKE 问题的**可行**解记作 $Y = (\mathbf{x}, z, r)$，这里对 $w \notin V$ 记 $\mathbf{x}[i, w] = 0$，这样 $\mathbf{x}$ 的大小可以统一为 $\mathbb{N}^{[-L..L]\times U}$。若 $\mathbf{x}$ 符合 K 的解的要求，z 为对应的 z_x，r 为对应的 r_x，则称 Y 是 K 的 (一个) **准确**解。

对于 $V \subset U$，称

$$K' = (V, \mathcal{Q}'_w, S', q)$$

$$\forall w \in V, \quad \mathcal{Q}'_w = \mathcal{Q}_w$$

$$\forall i \in \mathbb{Z}, \quad S'[i] = S[i] \cap V$$

为 $K = (U, \mathcal{Q}_w, S, q)$ 在 V 上的**限制**，记为 $K|_V$。

给定 $K|_V$ 与它的可行解 $Y_V = (\mathbf{x}, z, r)$，称

$$K' = (U \backslash V, \mathcal{Q}'_w, S', q')$$

$$\forall w \in U \backslash V, \quad \mathcal{Q}'_w = \mathcal{Q}_w$$

$$\forall i \in \mathbb{Z}, \quad S'[i] = S[z[i]] \backslash V$$

$$\forall i \in \mathbb{Z}, q'[i] = r[i]$$

为 $K = (U, \mathcal{Q}_w, S, q)$ 以 V 进行的**部分更新**，记为 $K^{V \leftarrow Y_V}$。

对于 $V \sqcup V' \subset U$ 的两个集合 V、V' 与 $K|_V$ 的可行解 $Y_V = (\mathbf{x}, z, r)$、$K^{V \leftarrow Y_V}|_{V'}$ 的可行解 $Y_{V'} = (\mathbf{x}', z', r')$，定义两个可行解的**组合**为 $K|_{V \sqcup V'}$ 的可行解

$$Y_{V'} \circ Y_V = (\mathbf{x}'', z'', r')$$

$$z''[i] = z[z'[i]], \quad \mathbf{x}''[i] = \mathbf{x}'[i] + \mathbf{x}[z'[i]]$$

这事实上是某种类似半直积的结构，由此可以验证组合具有结合律。值得注意的是，第二个式子可以写成

$$\mathbf{x}''[i, w] = \mathbf{x}'[i, w] + \mathbf{x}[z'[i], w]$$

于是当 $w \in V$ 时只有第二部分非零，当 $w \in V'$ 时只有第一部分非零，这里的加号事实上是某种**拼接**，而非真正的相加。

对于 $V \sqcup V' \subset U$ 的两个集合 V、V'，若 Y_V 是 $K|_V$ 的准确解、$Y_{V'}$ 是 $K^{V \leftarrow Y_V}|_{V'}$ 的准确解，则 $Y_{V'} \circ Y_V$ 是 $K|_{V \sqcup V'}$ 的准确解 [引理 4.5]。

与之前用 HKE 问题构造 HDP 更新的证明过程类似，这需要一些对解 $(\mathbf{x}, z, r)$ 结构的细致分析，并利用部分更新后的性质进行分类讨论。

4.3 分割求解

有分解与组合的方法后，我们就可以先进行部分的求解，再拼合成原问题的解了，下设原问题为 $K = (U, \mathcal{Q}_w, S, q)$。由于每个 $S[i]$ 支集大小不超过 b，所有 $S[i]$ 支集的并不超过 $\mathcal{O}(bL)$，于是取

$$r = [b/\log(4L+2)]$$

可在 $\mathcal{O}(Lr \log L \log r + (bL)\log r)$ 的时间内构造映射 $\mathcal{C} : [-L..L] \to [r]$ 使得

$$\forall i \in [-L..L], \quad \forall c \in [r], \quad |S[i] \cap \mathcal{C}^{-1}(c)| \le \mathcal{O}(\log L)$$

记 $U_c = \mathcal{C}^{-1}(c)$，定义 $K_1 = K$，对 c 从 1 到 r 执行：

- 求解 $K_c|_{U_c}$ 得到准确解 Y_c；
- 更新 $K_{c+1} = K_c^{U_c \leftarrow Y_c}$。

并设 $Y = Y_r \circ Y_{r-1} \circ \cdots \circ Y_1$，利用 4.2 节可知其的确为 K 的一个准确解。

由于 $U = U_1 \sqcup U_2 \sqcup \cdots \sqcup U_r$，可以发现进行每个部分的计算时只需要利用 $S[i]$ 在其中的部分，从而更新过程无需真的对 S 进行复制与修改。

由于共进行了 r 次求解，每次均长为 L、规模 $\mathcal{O}(\log L)$，求解部分的总复杂度为 $\mathcal{O}(rL\log^3 L)$。分解与组合的过程事实上不会超过此复杂度，从而再与染色复杂度结合，代入 r 的表达式即可最终得到求解总复杂度 $\mathcal{O}(bL\log^2 L)$、染色最终复杂度 $\mathcal{O}(bL\log r)$，求和得结论。

5 染色划分 II

上一部分，我们将 HKE 问题也缩减了规模，从而只需考虑 b 较小——准确来说，是 $b = \mathcal{O}(\log L)$ 时的解法，且我们需要此时的时间复杂度为 $\mathcal{O}(L\log^3 L)$ 量级。

但是，由于 b 仍然具有很大的随机性，即使已知 b 很小，直接求解也是困难的，除非能划归为最基本的情况，也就是 $b = 1$ 时，此时的 HKE 问题称为 **SHKE 问题** (S 指 Singleton)。进行完最后一次划分后，SHKE 的算法是可以直接写出的，这样就完成了证明。

5.1 完全分离染色

与上一部分完全类似，我们需要通过染色进行问题的分解。此时的染色需要做到**完全分离**，由引理：给定 m 个集合 $S_1, \ldots, S_m$ 满足

$$\forall i \in [m], \quad S_i \subset [n], \quad |S_i| \le b$$

则存在 $k \le \log(2m)$ 个映射 $\mathcal{C}_1, \ldots, \mathcal{C}_k : [n] \to [b^2]$ 使得

$$\forall i \in [m], \quad \exists j \in [k], \quad |\mathcal{C}_j(S_i)| = |S_i|$$

且构造时间复杂度为 $\mathcal{O}(n\log m+mb^3)$ [引理 4.10]。

由于有限集合上 $|\mathcal{C}_j(S_i)|=|S_i|$ 即代表单射，这就说明第 j 种染色方式完全分离了 S_i。与 4.1 中的证明类似，我们需要先证明二分时的结论，即对上述 S_i，可以在 $\mathcal{O}(n+mb^3)$ 的复杂度构造出一个 $\mathcal{C}:[n]\to[b^2]$，使得**至少有一半**的 S_i 被 h 完全分离 [命题 B.5]。

自然，有了这样的构造方式后，不断对剩余的 S_i 考虑并构造对应的 $\mathcal{C}$，即能得到结论。此命题的证明需要一些**概率方法**，具体来说，先考虑 $h:[n]\to[b^2]$ 为均匀随机的结果，则它将一个 S_i 完全分离的概率至少为 (S_i 的任何两个不同元素有 b^{-2} 的概率相同，利用概率的基本性质可得)

$$1-\frac{|S_i||S_i-1|}{2}\frac{1}{b^2}>\frac{1}{2}$$

再将此随机算法转化为决定算法即可。

5.2 最值性质

同样，在如此完全分离后，为了能够组合出正确结果，需要新的组合方式，这就依赖结果的**最值**性质。

对 $K_1=(U,\mathcal{Q}_w,S_1,q_1)$、$K_2=(U,\mathcal{Q}_w,S_2,q_2)$，定义它们的**上限问题**为

$$\max(K_1,K_2)=(U,\mathcal{Q}_w,S_3,q_3)$$

$$\forall i\in\mathbb{Z},\quad (S_3[i],q_3[i])=\begin{cases}(S_1[i],q_1[i]) & q_1[i]>q_2[i]\\ (S_2[i],q_2[i]) & q_1[i]<q_2[i]\\ (S_1[i]\cap S_2[i],q_1[i]) & q_1[i]=q_2[i]\end{cases}$$

对 K_1 的可行解 $Y_1=(\mathbf{x}_1,z_1,r_1)$、$K_2$ 的可行解 $Y_2=(\mathbf{x}_2,z_2,r_2)$，定义它们的**上限解**为上限问题 $\max(K_1,K_2)=(U,\mathcal{Q}_w,S_3,q_3)$ 的可行解

$$\max(Y_1,Y_2)=(\mathbf{x_3},z_3,r_3)$$

$$\forall i\in\mathbb{Z},\quad (\mathbf{x}_3[i],z_3[i])=\begin{cases}(\mathbf{x}_1[i],z_1[i]) & r_1[i]>r_2[i]\\ (\mathbf{x}_2[i],z_2[i]) & r_1[i]\le r_2[i]\end{cases},\quad r_3[i]=q_3[z_3[i]]+\sum_{w\in U}\mathcal{Q}_w(\mathbf{x}_3[i,w])$$

最值性质即，若 Y_1 是 K_1 的准确解，Y_2 是 K_2 的准确解，则 $\max(Y_1,Y_2)$ 是 $\max(K_1,K_2)$ 的准确解 [引理 4.7]。

此性质的证明比起 4.2 节的部分更新要简单一些，因为不涉及集合的改变，只须根据定义逐个 i 的考虑即可。不过，由于会存在是否松弛的情况，仍然需要一定的分类讨论。

5.3 双重分割

本节中，假设对长度为 L 的 SHKE 问题已有复杂度为 $\mathcal{O}(L+L_1\log L)$ 的算法，这里 L_1 为 $S[i]\neq\varnothing$ 的 i 的个数 [引理 4.1]，我们来通过分割进行最终的构造，下设原问题为 $K=(U,\mathcal{Q}_w,S,q)$。

利用完全分离染色的引理，可在 $\mathcal{O}((bL)\log L+Lb^3)$ 的时间内 (这里 bL 仍然源于所有 $S[i]$ 支集之并) 构造 $k=\mathcal{O}(\log L)$ 个映射 $\mathcal{C}_1,\dots,\mathcal{C}_k:U\to[b^2]$ 使得

$$\forall i\in[-L..L],\quad \exists j\in[k],\quad |\mathcal{C}_j(S[i])|=|S[i]|$$

对每个 $i \in [-L..L]$，将第一个使得上述条件成立的 j 记为 j_i，并对 $j \in [k]$ 定义

$$I^{(j)} = \{i \in [-L..L] \mid j_i = j\}$$

由定义可发现 $[-L..L] = I^{(1)} \sqcup I^{(2)} \sqcup \cdots \sqcup I^{(k)}$。进一步记

$$K^{(j)} = (U, \mathcal{Q}_w, S^{(j)}, q^{(j)})$$

$$S^{(j)}[i] = \begin{cases} S[i] & i \in I^{(j)} \\ \varnothing & i \notin I^{(j)} \end{cases}, \quad q^{(j)}[i] = \begin{cases} q[i] & i \in I^{(j)} \\ -\infty & i \notin I^{(j)} \end{cases}$$

由定义 $K = \max(K^{(1)}, \ldots, K^{(k)})$，于是只需求解每个 $K^{(j)}$ 即可构造出整体的解。直接利用定义计算即可看到，计算一次 max 的时间复杂度不会超过 $\mathcal{O}(bL)$，于是从每个 $K^{(j)}$ 的解构造整体解的时间复杂度为 $\mathcal{O}(kbL) = \mathcal{O}(Lb \log L)$。

最后，为了求解 $K^{(j)}$，完全类似 4.3 节进行**逐步分割组合**：记 $U_c = \mathcal{C}_j^{-1}(c)$，定义 $K_1 = K^{(j)}$，对 c 从 1 到 b^2 执行：

- 求解 $K_c|_{U_c}$ 得到准确解 Y_c；
- 更新 $K_{c+1} = K_c^{U_c \leftarrow Y_c}$。

利用定义，这里的每步求解的子问题中 $|S[i]| \le 1$ 均成立，从而是一个 SHKE 问题。根据假设，其存在 $\mathcal{O}(L + |I^{(j)}| \log L)$ 时间的算法，由此估算求解与组合的总复杂度不超过

$$\sum_{j=1}^{k} b^2 \mathcal{O}(L + |I^{(j)}| \log L) = \mathcal{O}(b^2(kL + \log L)) = \mathcal{O}(Lb^2 \log L)$$

由此，对每部分时间复杂度求和可知总时间复杂度的确达到了上节中 $\mathcal{O}(Lb^2(b + \log L))$ 的要求。

6 SHKE 求解

通过四步划分，我们终于将问题化归为了 SHKE 问题，也即规模为 1，每个 $S[i]$ 至多一个元素的 HKE 问题。仍记

$$L_1 = |\{i \in [-L..L] \mid S[i] \neq \varnothing\}|$$

我们将直接写出其复杂度为 $\mathcal{O}(L + L_1 \log L)$ 的算法，于是一步步回溯可得到整体的复杂度证明。

6.1 SMAWK 算法

为了构造出算法，一个非常重要的准备工作是由 Aggarwal、Klawe、Moran、Shor 与 Wilber 发明的 SMAWK 算法。

考虑 $m \times n$ 矩阵 $\mathbf{a} \in \underline{\mathbb{Z}}^{m \times n}$，且满足：

- **凸 Monge 性**

$$\forall 1 \le i < i' \le m, 1 \le j < j' \le n, \quad \mathbf{a}[i,j] + \mathbf{a}[i',j'] \ge \mathbf{a}[i,j'] + \mathbf{a}[i',j]$$

- **反向下阶梯形**

$$\forall 1 \le i \le i' \le m, 1 \le j \le j' \le n, \quad \mathbf{a}[i,j'] > -\infty \Longrightarrow \mathbf{a}[i',j] > -\infty$$

则 SMAWK 算法可通过 $\mathcal{O}(n(1+\log\lceil m/n\rceil))$ 的时间复杂度找到其每行的最大值所在位置 [定理 A.1]。

更严谨来说，可以利用定义验证一个反向下阶梯形的凸 Monge 矩阵满足**凸完全单调性**

$$\forall 1\le i<i'\le m, 1\le j<j'\le n,\quad \mathbf{a}[i,j]<\mathbf{a}[i,j']\Longrightarrow \mathbf{a}[i',j]<\mathbf{a}[i',j']$$

假设第 i 行最左侧的最大值位置为 $\mathbf{a}[i,j_i]$，由此性质可验证 j_i 随 i 单调增。由此，SMAWK 算法的最终输出为整数

$$1=r_1\le r_2\le\cdots\le r_{n+1}=m+1$$

使得

$$\forall i\in[r_j..(r_{j+1}-1)],\quad j_i=j$$

为介绍其作用，我们下面证明 2.3 节提到的引理，其核心思路与 3.3 节相似，由此可再次看出无松弛条件的 HKE 问题与更新 DP 表等价。回顾其表述为，设 $I\subset[n]$，$J\subset\bar{G}$ 或 $J\subset G$ 使得 $I\cap J=\varnothing$，且 $\mathrm{supp}\mathfrak{W}(J)=\{w\}$。假设已知对所有 $I\cup J$-最优交换对 (A',B') 均有 $|W(A')-W(B')|\le L'$，则可在 $\mathcal{O}(L+L'+|J|+w_{\max}\log w_{\max})$ 的时间复杂度将一个 I-最优的长为 L 的 DP 表更新为一个 $I\cup J$-最优的长为 L' 的 DP 表 [引理 2.7]。

不妨设 $J\subset\bar{G}$，另一种情况相似证明即可。设 J 中物品按价值从大到小排序为 $j_1,\dots,j_{|J|}$，定义

$$\mathcal{Q}(x)=\begin{cases}\sum_{t=1}^{x}p_{j_t} & x\in[0..|J|]\\ \mathcal{Q}(|J|) & x>|J|\end{cases}$$

由定义可发现其为满足 $\mathcal{Q}(0)=0$ 的非负凹函数。更新 DP 表的过程可以改写为

$$q'[z]=\max_{x\ge0}\big(q[z-xw]+\mathcal{Q}(x)\big)$$

由条件事实上有上界 $x\le(L+L')/w$。

将其拆分为对每个模 w 同余类的**最大和卷积问题**，向量 $a[i]$ 对应 q 中与 z 模 w 同余的分量，$b[i]$ 对应 $\mathcal{Q}(0)$ 到 $\mathcal{Q}([(L+L')/w])$。记

$$\mathbf{c}[i,j]=\begin{cases}a[j]+b[i-j] & j\le i\\ -\infty & j>i\end{cases}$$

可由 $\mathcal{Q}$ 的凹性验证矩阵符合要求，于是直接对其应用 SMAWK 算法可由定义得到结果。

现在我们进行复杂度的分析。由于只要知道了 a、b，$\mathbf{c}$ 的每个元素即可以通过常数时间访问，因此对每个同余类求解过程的总复杂度为 $\mathcal{O}((L+L')/w+1)$，于是乘同余类个数 w 得到用 SMAWK 算法求解的总复杂度

$$\mathcal{O}(L+L'+w)$$

较复杂的部分是对 $\mathcal{Q}$ 的预处理。利用位序的性质，只需考虑 j_1 到 $j_{2w_{\max}}$ 即必然能得到最优解，而利用线性时间中位数算法可在 $\mathcal{O}(|J|)$ 时间找到它们，进一步在 $\mathcal{O}(w_{\max}\log w_{\max})$ 时间排序后得到最终的总复杂度

$$\mathcal{O}(L+L'+w)+\mathcal{O}(|J|)+\mathcal{O}(w_{\max}\log w_{\max})$$

合并即得结论。

6.2 更新候选

利用 SMAWK 算法，我们也可以得到 SHKE 问题的一个快速算法，分为两部分实现。下设原问题为 $K = (U, \mathcal{Q}_w, S, q)$。首先，我们将通过与 6.1 节完全类似的技巧初步更新候选。

定义行数充分大、列数为 5 的矩阵 $\mathbf{p}$，初始有效行数为 0。下面所说的**插入** $(j; c, w, k, l)$ 是指，将当前有效行数加 1，并把对应的行五个元素改为 j, r, w, k, l。

对每个 $w \in [w_{\max}]$ 与模 w 余数 $c \in [0..(w-1)]$ 进行如下操作：记 $\equiv_w$ 代表模 w 同余，定义

$$I = \{j \in [-L..L] \mid j \equiv_w c\}$$

$$J = \{j \in [-L..L] \mid w \in S[j], j \equiv_w c\}$$

考虑矩阵 $\mathbf{a} \in \underline{\mathbb{Z}}^{I \times J}$，其中

$$\mathbf{a}[i,j] = \begin{cases} q[j] + \mathcal{Q}_w((i-j)/w) & i \geq j \\ 0 & i < j \end{cases}$$

由 $\mathcal{Q}$ 的凹性可对矩阵 $\mathbf{a}$ 应用 SMAWK 算法，对每个 $j \in J$，记

$$P_j = \{i \in [-L..L] \mid \forall 1 \leq j' < j \leq j'' \leq |J|, \mathbf{a}[i,j'] < \mathbf{a}[i,j], \mathbf{a}[i,j''] \leq \mathbf{a}[i,j]\}$$

即 P_j 中元素为使得 j 为第 i 行最左侧的最大值的 i。由 6.1 节中已介绍的性质，这样的 i 必然为 I 中的连续元素，也即事实上构成一个公差为 w 的**等差数列**。为使后续操作有意义，我们进行**规范**化 $\bar{P}_j = P_j \cap \{i \mid i > j\}$，则规范化后其若非空，则仍构成一个等差数列，由模 w 余数为 c 可设 (这里 $k \leq l$，且均为整数)

$$\bar{P}_j = \{c + kw, c + (k+1)w, \ldots, c + lw\}$$

对非空的 $\bar{P}_j$，将 $(j; c, w, k, l)$ 插入 $\mathbf{p}$，作为候选。

这样的候选构造意义与 6.1 节中几乎是一致的，不同的是，由于有了松弛条件，原本需要在 $\underline{\mathbb{Z}}^{I \times I}$ 上进行的 SMAWK 算法变为了在 $\underline{\mathbb{Z}}^{I \times J}$ 上，这就大大降低了复杂度。上述算法中，虽然看似有三重循环，但事实上只需对每个 $S[j]$ 进行一次遍历 (时间复杂度 $\mathcal{O}(L)$)，在 $S[j]$ 非空时取出 w、计算 r，进行操作即可，于是实际需要的循环次数不超过 L_1 次。这里 “不超过” 是由于不同的 j 可能对应相同的 w, r，我们只对**不重复**的 w, r 进行。下面考虑每次循环的时间复杂度。，利用 SMAWK 算法的结论，输出最大值所需时间为 (由 $|I| \leq (2L+1)/w + 1$，$|J| \geq 1$)

$$\mathcal{O}(|J|(1 + \log(|I|/|J|))) = \mathcal{O}(|J|(1 + \log(L/w))) = \mathcal{O}(|J| \log L)$$

而其他部分时间不会超过计算最大值，由此总时间复杂度

$$\mathcal{O}(L) + \sum_{|J| \neq \varnothing} \mathcal{O}(|J| \log L)$$

由于最多有 L_1 个 j 使得存在 $w \in S[j]$，且由不重复性同一个 j 不可能出现在两个不同的 J 中，因此

$$\sum_{|J| \neq \varnothing} |J| = L_1$$

这就得到了更新候选部分的总时间复杂度为

$$\mathcal{O}(L + L_1 \log L)$$

另一方面，由于每个 $j \in J$ 至多贡献一个 $\mathbf{p}$ 的行，我们可以知道 $\mathbf{p}$ 中至多被插入了 L_1 个元素，于是将行数设置为 L_1 即可。此估算将在 6.3 节计算时间复杂度时使用。

6.3 最终筛选

经过上一部分算法，我们得到了一个矩阵 $\mathbf{p}$，里面记录着一系列等差数列。下面，构造 $2L+1$ 个“桶”

$$\forall i \in [-L..L], \quad \mathbf{b}_i \in \mathbb{Z}^{L_1 \times 5}$$

用于确定**最终有效的候选**，这里 $\mathbf{b}_i$ 的第一个维度只需充分大即可，将所有 $\mathbf{b}_i$ 的初始有效行数都设为 0，接下来所说的插入与 6.2 节含义相同。

首先，对每个 $\mathbf{p}$ 中的 $(j;c,w,k,l)$，将其插入桶 $\mathbf{b}_{r+kw}$。接下来，对每个 i，初始化 $\mathbf{x}[i,\tilde{w}]$ 全为 0、$z[i]=i$、$r[i]=q[i]$。若 $\mathbf{b}_i$ 非空，计算其中所有 $(j;c,w,k,l)$ 的

$$r_j[i] = q[j] + \mathcal{Q}_w\left(\frac{i-j}{w}\right)$$

并将最优的 j 记为 j^*，对应的元素为 $(j^*;c^*,w^*,k^*,l^*)$。

若 $r_{j^*}[i] > r[i]$，说明可以进行更新，新的 $r[i]=r_j[i]$、$z[i]=j$，而 $\mathbf{x}[i,w]$ 当 $w \neq w^*$ 时为 0，否则为 $\frac{i-j}{w^*}$。此外，若 $i+w^* \leq c^*+l^*w^*$，可以产生新的候选，将 $(j^*;c^*,w^*,k^*,l^*)$ 插入桶 $\mathbf{b}_{i+w^*}$。

在对 $i \in [-L..L]$ 执行上述过程后，最终的 $(\mathbf{x},z,r)$ 即为解。此过程的正确性分析事实上与 6.1 节中证明的引理类似，加入松弛条件后只需要考虑更简单的情况，从而能进行最终的候选更新。为估算复杂度，考虑所有桶中出现过的元素 (计重数) 个数。由于在开始时执行了至多 L_1 次插入，之后对每个 i 至多进行一次插入，最终的插入次数为 $\mathcal{O}(L+L_1)=\mathcal{O}(L)$。而由于计算最优的 j 只需所有元素遍历一次，需要的时间复杂度与总插入次数相同，为 $\mathcal{O}(L)$。由此，第二部分的算法总复杂度为 $\mathcal{O}(L)$，最终复杂度

$$\mathcal{O}(L_1 \log L) + \mathcal{O}(L) = \mathcal{O}(L + L_1 \log L)$$

这就符合了我们对 SHKE 问题的复杂度要求。

7 总结与讨论

通过上面的层层划分与求解，我们最终得到了背包问题的一个 $w_{\max}$ 指数上已几乎为理论最优的

$$\mathcal{O}(n + w_{\max}^2 \log^4 w_{\max})$$

时间复杂度的算法。

这个过程看起来虽然繁琐，但其实十分优雅：每一步都在进行规模的削减，也化为了更加特殊的子问题与求解。总体来说，最核心的思路仍然是动态规划，与进行了时空权衡的**带提示集动态规划**问题，而基础算法也仍然是 SMAWK 算法。

最后，我们将讨论原论文可能的后续工作与优化方向：

- 第一步的问题分解中，求解 I_{W_1}-最优的 DP 表需要花费 $\mathcal{O}(n + w_{\max}^2 \log^4 w_{\max})$，而后续只需要 $\mathcal{O}(n + w_{\max}^2 \log^{1.5} w_{\max})$。若能进行更精细的划分以具有稍特殊一些的性质，或许能平衡这两部分的复杂度，以降低总体复杂度。

- SHKE 问题的 $\mathcal{O}(L + L_1 \log L)$ 算法已经几乎最优了，因为无论如何 $\mathcal{O}(L)$ 的遍历时间是需要的，而由于仍然存在不同更新方式的对比，$\mathcal{O}(L)$ 的算法也不太可能出现。不过，原论文并没有利用决策空间与排除的方法进行更细致的理论分析以证明这件事，个人认为是可以证明的。

- 对于一般的 HKE 问题，进行染色化归确实是一种较好的方式，但的确可能存在其他方法降低复杂度——染色的过程事实上需要较高复杂度。因此，对其理论复杂度的分析与实际算法的进一步开发都是需要的。对于 DP 表的研究远比对于 HDP 表的研究要丰富，因此想要更好求解 HKE 问题，也需要对 HDP 表的结构有进一步的刻画。
- 最后，HDP 表的引入也可以带来更多近似算法的设计。若我们允许一定的误差，可以存在近似更新 HDP 表的过程，而其与 DP 表不同的权衡特性或许能设计出不同的近似算法。

参考文献

[1] JIN C. 0-1 knapsack in nearly quadratic time[C/OL]//STOC 2024: Proceedings of the 56th Annual ACM Symposium on Theory of Computing. New York, NY, USA: Association for Computing Machinery, 2024: 271–282. https://doi.org/10.1145/3618260.3649618.

[2] CHEN L, LIAN J, MAO Y, et al. A nearly quadratic-time fptas for knapsack[C/OL]//STOC 2024: Proceedings of the 56th Annual ACM Symposium on Theory of Computing. New York, NY, USA: Association for Computing Machinery, 2024: 283–294. https://doi.org/10.1145/3618260.3649730.

[3] BRINGMANN K. Knapsack with small items in near-quadratic time[C/OL]//STOC 2024: Proceedings of the 56th Annual ACM Symposium on Theory of Computing. New York, NY, USA: Association for Computing Machinery, 2024: 259–270. https://doi.org/10.1145/3618260.3649719.

[4] MAO X. $(1-\varepsilon)$-approximation of knapsack in nearly quadratic time[C/OL]//STOC 2024: Proceedings of the 56th Annual ACM Symposium on Theory of Computing. New York, NY, USA: Association for Computing Machinery, 2024: 295–306. https://doi.org/10.1145/3618260.3649677.

[5] JIN C. 0-1 knapsack in nearly quadratic time[A/OL]. 2024. arXiv: 2308.04093. https://arxiv.org/abs/2308.04093.

[6] CYGAN M, MUCHA M, WEGRZYCKI K, et al. On problems equivalent to (min, +)-convolution [J/OL]. CoRR, 2017, abs/1702.07669. http://arxiv.org/abs/1702.07669.