0-1 Knapsack in Nearly Quadratic Time

Ce Jin*
MIT

August 9, 2023

Abstract

We study pseudo-polynomial time algorithms for the fundamental *0-1 Knapsack* problem. Recent research interest has focused on its fine-grained complexity with respect to the number of items n and the *maximum item weight* $w_{\max}$. Under $(\min,+)$-convolution hypothesis, 0-1 Knapsack does not have $O((n+w_{\max})^{2-\delta})$ time algorithms (Cygan-Mucha-Węgrzycki-Włodarczyk 2017 and Künnemann-Paturi-Schneider 2017). On the upper bound side, currently the fastest algorithm runs in $\widetilde{O}(n+w_{\max}^{12/5})$ time (Chen, Lian, Mao, and Zhang 2023), improving the earlier $O(n+w_{\max}^3)$-time algorithm by Polak, Rohwedder, and Węgrzycki (2021).

In this paper, we close this gap between the upper bound and the conditional lower bound (up to subpolynomial factors):

- The 0-1 Knapsack problem has a deterministic algorithm in $O(n+w_{\max}^2\log^4 w_{\max})$ time.

Our algorithm combines and extends several recent structural results and algorithmic techniques from the literature on knapsack-type problems:

1. We generalize the "fine-grained proximity" technique of Chen, Lian, Mao, and Zhang (2023) derived from the additive-combinatorial results of Bringmann and Wellnitz (2021) on dense subset sums. This allows us to bound the support size of the useful partial solutions in the dynamic program.
2. To exploit the small support size, our main technical component is a vast extension of the "witness propagation" method, originally designed by Deng, Mao, and Zhong (2023) for speeding up dynamic programming in the easier unbounded knapsack settings. To extend this approach to our 0-1 setting, we use a novel pruning method, as well as the two-level color-coding of Bringmann (2017) and the SMAWK algorithm on tall matrices.

*cejin@mit.edu. Supported by NSF CCF-2129139, CCF-2127597, and a Siebel Scholarship.

Contents

1 Introduction

In the *0-1 Knapsack* problem, we are given a knapsack capacity $t \in \mathbb{Z}^+$ and n items $(w_1, p_1), \ldots, (w_n, p_n)$, where $w_i, p_i \in \mathbb{Z}^+$ denote the *weight* and *profit* of the i-th item, and we want to select a subset $X \subseteq [n]$ of items satisfying the capacity constraint $W(X) := \sum_{i \in X} w_i \leq t$, while maximizing the total profit $P(X) := \sum_{i \in X} p_i$.

Knapsack is a fundamental problem in computer science.[1] It is among Karp's 21 NP-complete problems [Kar72], and the fastest known algorithm runs in $O(2^{n/2})$ time [HS74, SS81]. However, when the input integers are small, it is more preferable to use *pseudopolynomial time* algorithms that have polynomial time dependence on both n and the input integers. Our work focuses on this pseudopolynomial regime. A well-known example of pseudopolynomial algorithms is the textbook $O(nt)$-time Dynamic Programming (DP) algorithm for Knapsack, given by Bellmann [Bel57] in 1957. Finding faster pseudopolynomial algorithms for Knapsack became an important topic in combinatorial optimization and operation research; see the book of Kellerer, Pferschy, and Pisinger [KPP04] for a nice summary of the results known by the beginning of this century. In the last few years, research on Knapsack (and the easier Subset Sum problem, which is the special case of Knapsack where $p_i = w_i$) has been revived by recent developments in fine-grained complexity (e.g, [CMWW19, KPS17, KX19, Bri17, BHSS18, ABHS22b]) and integer programming (e.g., [EW20, PRW21]), and the central question is to understand the best possible time complexities for solving these knapsack-type problems.

Cygan, Mucha, Węgrzycki, and Włodarczyk [CMWW19] and Künnemann, Paturi, and Schneider [KPS17] showed that the $O(nt)$ time complexity for Knapsack is essentially optimal (in the regime of $t = \Theta(n)$) under the $(\min, +)$-convolution hypothesis. To cope with this hardness result, recent interest has focused on parameterizing the running time in terms of n and the *maximum item weight* $w_{\max}$ (or the *maximum item profit* $p_{\max}$), instead of the knapsack capacity t. This would be useful when the item weights are much smaller than the capacity, and results along this line would offer us a more fine-grained understanding of knapsack-type problems. This parameterization is also natural from the perspective of integer linear programming (e.g., [EW20]): when formulating Knapsack as an integer linear program, the maximum item weight $w_{\max}$ corresponds to the standard parameter Δ, maximum absolute value in the input matrix.

However, despite extensive research on 0-1 Knapsack along these lines, our understanding about the dependence on $w_{\max}$ is still incomplete. Known fine-grained lower bounds only ruled out $(n + w_{\max})^{2-\delta}$ algorithms for Knapsack [CMWW19, KPS17] (for $\delta > 0$). In comparison, Bellman's dynamic programming algorithm only runs in $O(nt) \leq O(n^2 w_{\max})$ time. Several papers obtained the bound $\widetilde{O}(nw_{\max}^2)$ via various methods [EW20, BHSS18, AT19, KP04].[2] Polak, Rohwedder, and Węgrzycki [PRW21] carefully combined the *proximity technique* of Eisenbrand and Weismantel [EW20] from integer programming with the concave $(\max, +)$ convolution algorithm ([KP04] or [AKM+87]), and obtained an $O(n + w_{\max}^3)$ algorithm for Knapsack. These algorithms have cubic dependence on $(n + w_{\max})$. Finally, the very recent work by Chen, Lian, Mao, and Zhang [CLMZ23] broke this cubic barrier with an $\widetilde{O}(n + w_{\max}^{12/5})$-time algorithm, which was based on additive-combinatorial results of Bringmann and Wellnitz [BW21].[3]

[1] In this paper we use the term Knapsack to refer to 0-1 Knapsack (as opposed to other variants such as Unbounded Knapsack and Bounded Knapsack).

[2] We use $\widetilde{O}(f)$ to denote $O(f \operatorname{poly} \log f)$.

[3] An earlier work by Bringmann and Cassis [BC23] obtained an algorithm in $\widetilde{O}(nw_{\max}p_{\max}^{2/3})$ time, which was the

None of the above algorithms match the $(n+w_{\max})^{2-o(1)}$ conditional lower bound. The following question has been asked by [PRW21, BC22, CLMZ23]:

Main question: Can 0-1 Knapsack be solved in $\widetilde{O}(n+w_{\max}^2)$ time?

We remark that this $\widetilde{O}(n+w_{\max}^2)$ running time is known to be achievable for the easier *Unbounded Knapsack* problem (where each item has infinitely many copies available) [AT19, CH22, DMZ23], matching the $(n+w_{\max})^{2-\delta}$ conditional lower bound for Unbounded Knapsack [CMWW19, KPS17]. As argued by [PRW21], the 0-1 setting appears to be much more difficult, and most of the techniques for Unbounded Knapsack do not appear to apply to the 0-1 setting.

1.1 Our contribution

In this paper, we affirmatively resolve this main question, closing the gap between the previous $\widetilde{O}(n+w_{\max}^{12/5})$ upper bound [CLMZ23] and the quadratic conditional lower bound [CMWW19, KPS17].

Theorem 1.1. *The 0-1 Knapsack problem can be solved by a deterministic algorithm with time complexity $O(n+w_{\max}^2\log^4 w_{\max})$.*

In our paper we only describe an algorithm that outputs the total profit of the optimal knapsack solution. It can be modified to output an actual solution using the standard technique of back-pointers, without affecting the asymptotic time complexity.

By a reduction described in [PRW21, Section 4], we have the following corollary which parameterizes the running time by the largest item profit $p_{\max}$ instead of $w_{\max}$.

Corollary 1.2. *The 0-1 Knapsack problem can be solved by a deterministic algorithm with time complexity $O(n+p_{\max}^2\log^4 p_{\max})$.*

Chronological Remarks. The current paper is a substantially updated version of our earlier manuscript (posted on arXiv[4] in July 2023). This earlier manuscript contained much weaker results, and is obsolete now. Our current paper incorporates part of the techniques from our earlier manuscript, and also builds on the very recent work by Chen, Lian, Mao, and Zhang [CLMZ23] (arXiv, July 2023).

Independent works. Very recently, Bringmann [Bri23] (arXiv, August 6, 2023) also independently obtained an $\widetilde{O}(n+w_{\max}^2)$ time algorithm for 0-1 Knapsack (more generally, Bounded Knapsack).

1.2 Technical overview

Our Knapsack algorithm combines and extends several recent structural results and algorithmic techniques from the literature on knapsack-type problems. In particular, we crucially build on the techniques from two previous papers by Chen, Lian, Mao, and Zhang [CLMZ23], and by Deng, Mao, and Zhong [DMZ23]. Now we review the techniques in prior works and describe the ideas behind our improvement.

first algorithm for 0-1 Knapsack with subcubic dependence on $(n+w_{\max}+p_{\max})$.

[4] https://arxiv.org/abs/2307.09454

Fine-grained proximity based on additive combinatorics. There was a long line of work in the 80's on designing Subset Sum algorithms using techniques from *additive combinatorics* [GM91, CFG89, Cha99a, Fre90, Fre88, Cha99b], and more recently these techniques have been revived and applied to not only Subset Sum [KX19, MWW19, BW21, PRW21] but also the more difficult Knapsack problem [DJM23, CLMZ23]. Ultimately, these algorithms directly or indirectly rely on the following powerful result in additive combinatorics, pioneered by Freiman [Fre93] and Sárközy [Sár94] and tightened by Szemerédi and Vu [SV06], and more recently strengthened by Conlon, Fox, and Pham [CFP21]: Let $\mathcal{S}(A) = \{\sum_{b\in B} b : B \subseteq A\}$ denote the subset sums of A. Then, if set $A \subseteq [N]$ has size $|A| \gg \sqrt{N}$, then $\mathcal{S}(A)$ contains a (homogeneous) arithmetic progression of length N.

Another technique used in recent knapsack algorithms is the *proximity technique* from the integer programming literature, see e.g., [CGST86, EW20]. When specialized to the Knapsack case (1-dimensional integer linear program), a proximity result refers to a distance upper bound between the optimal knapsack solution and the *greedy solution* (sort items in decreasing order of efficiencies p_i/w_i, and take the maximal prefix without violating the capacity constraint). Polak, Rohwedder, and Węgrzycki [PRW21] exploited the fact that these two solutions differ by at most $O(w_{\max})$ items, which allowed them to shrink the size of the dynamic programming (DP) table from t down to $O(w_{\max}^2)$ (by performing DP on top of the greedy solution to find an optimal exchange solution). They achieved $O(n + w_{\max}^3)$ time by batch-updating items of the same weight w using the SMAWK algorithm [AKM+87] (see also [KP04, AT19]).

The very recent paper by Chen, Lian, Mao, and Zhang [CLMZ23] developed a new "fine-grained proximity" technique that combines these two lines of approach. They used the additive-combinatorial results of Bringmann and Wellnitz [BW21] (which built on works of Sárközy [Sár89, Sár94] and Galil and Margalit [GM91]) to obtain several powerful structural lemmas involving the support size of two multisets A, B (with integers from $[w_{\max}]$) that avoid non-zero common subset sums, and these structural lemmas were translated into proximity results using exchange arguments. These fine-grained proximity results of [CLMZ23] are more powerful than the earlier proximity bounds used in [PRW21, EW20]; the following lemma from [CLMZ23] is one example: given a Knapsack instance, we can partition the item weights into two subsets $[w_{\max}] = \mathcal{W}_1 \uplus \mathcal{W}_2$, such that $|\mathcal{W}_1| \le \widetilde{O}(\sqrt{w_{\max}})$, and the differing items between the greedy solution and the optimal solution whose weights belong to $\mathcal{W}_2$ can only have total weight $O(w_{\max}^{3/2})$. This lemma immediately led to a simple $\widetilde{O}(n + w_{\max}^{5/2})$ algorithm [CLMZ23]. A bottleneck step in this algorithm is to use DP to compute partial solutions consisting of items with weights from $\mathcal{W}_1$: they need to perform the batch DP update (based on SMAWK) $|\mathcal{W}_1|$ times, and the size of the DP table is still $O(w_{\max}^2)$ as in [PRW21], so the total time for this step is $\widetilde{O}(w_{\max}^{2.5})$. To overcome this bottleneck, [CLMZ23] used more refined proximity results based on the multiplicity of item weights, and obtained an improved running time $\widetilde{O}(n + w_{\max}^{2.4})$.

DP strategy based on multiplicity. In our work, we completely overcome this bottleneck of [CLMZ23]: we can implement the DP for items with weights from $\mathcal{W}_1$ in only $\widetilde{O}(w_{\max}^2)$ time. This is the main technical part of our paper. (The other bottleneck in [CLMZ23]'s simple $\widetilde{O}(n+w_{\max}^{5/2})$ time algorithm is to deal with items whose weights come from $\mathcal{W}_2$, but this part can be improved more easily by dividing into $O(\log w_{\max})$ partitions with smoothly changing parameters. See Section 3.1.) Now we give an overview of our improvement.

We rely on another additive-combinatorial lemma (Lemma 3.6) which can be derived from the

results of Bringmann and Wellnitz [BW21]; it is analogous and inspired by the fine-grained proximity results of [CLMZ23], but is not directly comparable to theirs. It implies the following proximity result: Let D denote the set of differing items between the greedy solution and the optimal solution. Then, for any $r \geq 1$, there can be at most $\widetilde{O}(\sqrt{w_{\max}/r})$ many weights $w \in [w_{\max}]$ such that D contains at least r items of weight w (i.e., w has multiplicity $\geq r$ in the item weights of D). In other words, if we figuratively think of the histogram of the weights of items in D, then the number of columns in the histogram with height $\geq r$ should be at most $\widetilde{O}(\sqrt{w_{\max}/r})$. As a corollary, the total area below height r in this histogram is at most $\widetilde{O}(\sqrt{rw_{\max}})$.

Our DP algorithm exploits the aforementioned structure of D as follows. We perform the DP in $O(\log w_{\max})$ phases, where in the j-th phase ($j \geq 1$) we update the current DP table with all items of *rank* in $[2^{j-1}, 2^j)$. Here, the *rank* of a weight-w item is defined as the rank of its profit among all weight-w items (an item with rank 1 is the most profitable item among its weight class). By the end of phase j, our DP table should contain the partial solution consisting of all items in D of rank $< 2^j$, i.e., the partial solution that corresponds to the part below height 2^j in the histogram representing D. As we mentioned earlier, this partial solution only has $\widetilde{O}(\sqrt{2^j w_{\max}})$ items, and hence $\widetilde{O}(w_{\max} \cdot \sqrt{2^j w_{\max}})$ total weight, so the size of the DP table at the end of phase j only needs to be $L_j := \widetilde{O}(w_{\max} \cdot \sqrt{2^j w_{\max}})$.

To efficiently implement the DP in each phase, we need to crucially exploit the aforementioned fact that the number of weights $w \in \mathcal{W}_1$ with multiplicity $\geq 2^{j-1} - 1$ in D is at most $b_j := \widetilde{O}(\sqrt{w_{\max}/2^j})$. (Note that in phase $j = 1$ this threshold is $2^{j-1} - 1 = 0$, and the upper bound $b_1 = \widetilde{O}(\sqrt{w_{\max}})$ simply follows from $|\mathcal{W}_1| = \widetilde{O}(\sqrt{w_{\max}})$ guaranteed by [CLMZ23]'s partition.) Our goal is to perform each phase of the DP updates in $\widetilde{O}(b_j \cdot L_j) = \widetilde{O}(w_{\max}^2)$ time. To achieve this goal, we surprisingly adapt a recent technique introduced in the much easier *unbounded* knapsack settings by Deng, Mao, and Zhong [DMZ23], called "witness propagation". In the following we briefly review this technique.

Transfer techniques from the unbounded setting. The *unbounded* knapsack/subset sum problems, where each item has infinitely many copies available, are usually easier for two main reasons: (1) Since there are infinite supply of items, we do not need to keep track of which items are used so far in the DP. (2) There are more powerful structural results available, in particular the Carathéodory-type theorems [ES06, Kle22, CH22, DMZ23] which show the existence of optimal solution vectors with only logarithmic support size.

Deng, Mao, Zhong [DMZ23] recently exploited the small support size to design near-optimal algorithms for several unbounded-knapsack-type problems, based on their key new technique termed "witness propagation". The idea is that, since the optimal solutions must have small support size (but possibly with high multiplicity), one can first prepare the "base solutions", which are partial solutions with small support and multiplicity at most one. Then, they gradually build full solutions from these base solutions, by "propagating the witnesses" (that is, increase the multiplicity of some item with non-zero multiplicity). The time complexity of this approach is low since the support sizes are small.

Now we come back to our DP framework for 0-1 knapsack described earlier, and observe that we are in a very similar situation to the unbounded knapsack setting of [DMZ23]. In our case, if we intuitively view our DP as gradually growing the columns of the histogram representing D, then after phase $j-1$, there can be only $\leq b_j$ columns in the histogram that may continue growing in subsequent phases. This means the "active support" of our partial solutions has size $\leq b_j$: when we

extend a partial solution in the DP table during phase j, we only need to consider items from b_j many weight classes, namely those weights that have "full multiplicity" in this partial solution by the end of phase $j-1$. (If there are more than b_j many such weights, then the proximity result implies that this partial solution cannot be extended to the optimal solution, and we can safely discard it.) This gives us hope of implementing the DP of each phase in $\widetilde{O}(b_j \cdot L_j) = \widetilde{O}(w_{\max}^2)$ using the witness propagation idea from [DMZ23].

However, we still need to overcome several difficulties that arise from the huge difference between 0-1 setting and unbounded setting. In particular, the convenient property (1) for unbounded knapsack mentioned above no longer applies to the 0-1 setting. In the following we briefly explain how we implement the witness propagation idea in the 0-1 setting.

Witness propagation in the 0-1 setting. In each phase j of our DP framework, we are faced with the following task (let $b = b_j, L = L_j$): we have a DP table $q[\,]$ of size L, and each entry $q[z]$ of the DP table is associated with a set $S[z] \subseteq \mathcal{W}_1$ of size $|S[z]| \le b$ (this is the "active support" of the partial solution corresponding to $q[z]$). For each entry $q[z]$ and each $w \in S[z]$, we would like to perform the DP update $q'[z+xw] \leftarrow \max(q'[z+xw], q[z] + Q_w(x))$ for all $x \ge 0$, where $Q_w(x)$ is the total profit of the top x remaining items of weight w (note that $Q_w(\cdot)$ is concave). We would like to compute the final DP table $q'[\,]$ in $\widetilde{O}(bL)$ time.

We first focus on an interesting case where each set $S[z]$ has size at most $b = 1$. Similarly to [PRW21, AT19, KP04], we try to use the SMAWK algorithm to perform these DP updates. However, since these sets $S[z]$ may contain different types of weights w, we need to deal with them separately. This means that for each weight w, there may be only sublinearly many indices z with $S[z] = \{w\}$. Hence, in order to save time, we need to do SMAWK for each w in time complexity sublinear in the entire DP table size L, and only near-linear in $n_w = \big|\{z : S[z] = \{w\}\}\big|$. So we need to let SMAWK return a compact output representation, which partitions the DP table into n_w segments, or more precisely, arithmetic progressions (APs) of difference w, where each AP contains the indices i for which $q'[i]$ is maximized by $q[z] + Q_w((i-z)/w)$ for the same $z \in \{z : S[z] = \{w\}\}$. This is an very interesting scenario where we actually need to use the tall-matrix version of SMAWK.

Then, we need to update these APs returned by these SMAWK algorithm invocations to the DP table $q'[\,]$. That is, for each i, we would like to pick the AP that contains i and maximizes the profit $q[z] + Q_w((i-z)/w)$ mentioned earlier. Naively going through each element in every AP would take time proportional to the total length of these APs. This would be too slow: although the total number of APs is only $O(L)$, their total length could still be very large. To solve this issue, we design a novel skipping technique, so that we can ignore suffixes of some of the APs, while still ensuring that we do not lose the optimal solution, so that the total time is reduced to $\widetilde{O}(L)$. Now we briefly describe this skipping technique:

We initialize an empty bucket $B[i]$ for each index i in the DP table. For each of the $O(L)$ many APs returned by SMAWK, we insert the (description of the) AP into the bucket indexed by the beginning element of this AP. Then we iterate over the buckets $B[i]$ in increasing order of i. For each $B[i]$, we pick the AP from this bucket that maximizes the profit value at i, and update the profit value $q'[i]$ accordingly. Then, we copy this AP from bucket $B[i]$ to the bucket indexed by the successor of i in this AP; the other APs in bucket $B[i]$ will not be copied. In this way, the total time is $O(L)$, since we start with $O(L)$ APs and each bucket only copies one AP to another bucket.

This skipping technique seemingly may omit some useful updates. However, we can show that it is actually fine, as it never loses the optimal solution. Roughly speaking, we can prove that,

whenever we omit some solution for $q'[i]$, there is always a better solution for $q'[i]$ that uses items of two different weights (this is shown using the concavity of $Q_w(\cdot)$). However, since $b = 1$, we know in advance that solutions with support size ≥ 2 cannot be optimal and can be safely discarded. This means our skipping technique never omits the optimal solution.

In the main text of the paper, we formalize the intuition above, and abstract out a core problem called HintedKnapsackExtend$^+$ (Problem 1) that captures the scenario described above in a more modular way. We prove some helper lemmas for Problem 1 that allows us to decompose an instance with large b to multiple instances with smaller b. Hence, having solved the case with $b = 1$, we can extend them to larger b by using the two-level color-coding technique originally used by Bringmann [Bri17] in his subset sum algorithm.

1.3 Further related works

In contrast to our 0-1 setting, the *unbounded* setting (where each item has infinitely many copies available) has also been widely studied in the literature of Knapsack and Subset Sum algorithms, e.g., [LPV20, JR19, JR22, AT19, CH22, Kle22, DMZ23].

For the easier Subset Sum problem, an early result for Subset Sum in terms of n and $w_{\max}$ is Pisinger's deterministic $O(nw_{\max})$-time algorithm for Subset Sum [Pis99]. This is not completely subsumed by Bringmann's $\widetilde{O}(n+t) \leq \widetilde{O}(nw_{\max})$ time algorithm [Bri17], due to the extra log factors and randomization in the latter result. More recently, Polak, Rohwedder, and Węgrzycki [PRW21] observed that an $\widetilde{O}(n + w_{\max}^2)$ time algorithm directly follows from combining their proximity technique with Bringmann's $O(n + t)$ Subset Sum algorithm [Bri17]. They improved it to $\widetilde{O}(n + w_{\max}^{5/3})$ time, by further incorporating additive combinatorial techniques by [BW21]. Very recently, [CLMZ23] obtained $\widetilde{O}(n + w_{\max}^{3/2})$-time algorithm for Subset Sum, using their fine-grained proximity technique based on additive combinatorial results of [BW21].

Recently there has also been a lot of work on approximation algorithms for Knapsack and Subset Sum (and Partition) [Cha18, Jin19, MWW19, BN21, DJM23]. For example, the fastest known $(1-\varepsilon)$ approximation algorithm for 0-1 Knapsack runs in $\widetilde{O}(n + 1/\varepsilon^{2.2})$ time [DJM23]. Notably, [DJM23] also used the additive combinatorial results of [BW21] to design knapsack approximation algorithms; this was the first application of additive combinatorial technique to knapsack algorithms.

1.4 Open problems

There are several interesting open questions.

- Can we $(1 - \varepsilon)$-approximate 0-1 Knapsack in $\widetilde{O}(n + \varepsilon^{-2})$ time? In the special case where all profits p_i are real values in $[1, 2]$, we can round p_i to integer multiples of ε, and use our $\widetilde{O}(n + p_{\max}^2)$-time algorithm to solve it in $\widetilde{O}(n + \varepsilon^{-2})$ time. We suspect the general case is also solvable by extending our techniques.

- In the regime where n is much smaller than $w_{\max}$, can we get faster algorithms for 0-1 Knapsack? Inspecting the algorithm in [CLMZ23, Section 4], its running time can also be bounded by $\widetilde{O}(nw_{\max}^{1.5})$. Can we achieve $\widetilde{O}(nw_{\max})$ time (which would also match the $(n + w_{\max})^{2-o(1)}$ conditional lower bound based on $(\min, +)$-convolution hypothesis [CMWW19, KPS17])?

- Can we solve 0-1 Knapsack in $O((n + w_{\max} + p_{\max})^{2-\delta})$ time for any constant $\delta > 0$? Bringmann and Cassis [BC22] gave algorithms of such running time for the easier unbounded knap-

sack problem. They also showed that such algorithms require computing bounded-difference $(\min,+)$-convolution [CL15, CDXZ22].

- Can we solve 0-1 Knapsack in $O(n + w_{\max}^2/2^{\Omega(\sqrt{\log w_{\max}})})$ time, matching the best known running time for $(\min,+)$-convolution [Wil18, BCD+14, CW21]? Algorithms with such running time are known for the easier unbounded knapsack problem [AT19, CH22, DMZ23].
- Can Subset Sum be solved in $\widetilde{O}(n+w_{\max})$ time? This question has been repeatedly asked in the literature [ABJ+19, ABHS22a, BW21, PRW21, BC22]. Currently the best result is the very recent $\widetilde{O}(n + w_{\max}^{3/2})$-time randomized algorithm by Chen, Lian, Mao, and Zhang [CLMZ23].
- Can our techniques be useful for other related problems, such as scheduling [BFH+22, ABHS22a, KPR23] or low-dimensional integer linear proramming [EW20]?

Paper organization

Section 2 contains definitions, notations, and some lemmas from previous works, which are essential for understanding Section 3. Then, in Section 3 we describe our algorithm for 0-1 Knapsack. A key subroutine of our algorithm is deferred to Section 4.

2 Preliminaries

2.1 Notations and definitions

We use $\widetilde{O}(f)$ to denote $O(f \operatorname{poly}\log f)$. Let $[N] = \{1, 2, \ldots, N\}$.

Multisets and subset sums. For an integer multiset X, and an integer x, we use $\mu_X(x)$ to denote the multiplicity of x in X. For a multiset X, the *support* of X is the set of elements it contains, denoted as $\operatorname{supp}(X) := \{x : \mu_X(x) \geq 1\}$. We say a multiset X is *supported on* $[N]$ if $\operatorname{supp}(X) \subseteq [N]$. For multisets A, B we say A is a subset of B (and write $A \subseteq B$) if for all $a \in A$, $\mu_B(a) \geq \mu_A(a)$. We write $A \uplus B$ as the union of A and B by adding multiplicities.

The *size* of a multiset X is $|X| = \sum_{x\in\mathbb{Z}} \mu_X(x)$, and the *sum of elements* in X is $\Sigma(X) = \sum_{x\in\mathbb{Z}} x \cdot \mu_X(x)$. The set of all *subset sums* of X is $\mathcal{S}(X) := \{\Sigma(Y) : Y \subseteq X\}$. We also define $\mathcal{S}^*(X) := \{\Sigma(Y) : Y \subseteq X, Y \neq \varnothing\}$ to be the set of subset sums formed by *non-empty* subsets of X.

The *r-support* of a multiset X is the set of items in X with multiplicity at least r, denoted as $\operatorname{supp}_r(X) := \{x : \mu_X(x) \geq r\}$.

Vectors and arrays. We will work with vectors in $\mathbb{Z}^{\mathcal{I}}$ where $\mathcal{I}$ is some index set. We sometimes denote vectors in boldface, e.g., $\boldsymbol{x} \in \mathbb{Z}^{\mathcal{I}}$, and use non-boldface with subscript to denote its coordinate, e.g., $x_i \in \mathbb{Z}$ (for $i \in \mathcal{I}$). Let $\operatorname{supp}(\boldsymbol{x}) := \{i \in \mathcal{I} : x_i \neq 0\}$, $\|\boldsymbol{x}\|_0 := |\operatorname{supp}(\boldsymbol{x})|$, and $\|\boldsymbol{x}\|_1 := \sum_{i\in\mathcal{I}} |x_i|$. Let $\mathbf{0}$ denote the zero vector. For $i \in \mathcal{I}$, let $\boldsymbol{e}_i$ denote the unit vector with i-th coordinate being 1 and the remaining coordinates being 0.

We use $A[\ell \mathinner{..} r]$ to denote an array indexed by integers $i \in \{\ell, \ell+1, \ldots, r\}$. The i-th entry of the array is $A[i]$. Sometimes we consider arrays of vectors, denoted by $\boldsymbol{x}[\ell \mathinner{..} r]$, in which every entry $\boldsymbol{x}[i] \in \mathbb{Z}^{\mathcal{I}}$ is a vector, and we use $x[i]_j$ to denote the j-th coordinate of the vector $\boldsymbol{x}[i]$ (for $j \in \mathcal{I}$).

0-1 Knapsack. In the 0-1 Knapsack problem with n input items $(w_1, p_1), \ldots, (w_n, p_n)$ (where *weights* $w_i \le w_{\max}$ and *profits* $p_i \le p_{\max}$ are positive integers) and knapsack capacity t, an *optimal knapsack solution* is an item subset $X \subseteq [n]$ that maximizes the total profit

$$P(X) := \sum_{i \in X} p_i, \tag{1}$$

subject to the capacity constraint

$$W(X) := \sum_{i \in X} w_i \le t. \tag{2}$$

We will frequently use the following notations:

- Let $\mathcal{W} = \mathrm{supp}(\{w_1, w_2, \ldots, w_n\}) \subseteq [w_{\max}]$ be the set of input item weights.
- For $\mathcal{W}' \subseteq \mathcal{W}$, let $I_{\mathcal{W}'} := \{i \in [n] : w_i \in \mathcal{W}'\}$ denote the set of items with weights in $\mathcal{W}'$.
- For $I = \{i_1, \ldots, i_{|I|}\} \subseteq [n]$, let $\mathrm{weights}(I) = \{w_{i_1}, \ldots, w_{i_{|I|}}\}$ be the *multiset* of weights of items in I.

We assume $w_{\max} \le t$ by ignoring items that are too large to fit into the knapsack. We assume $w_1 + \cdots + w_n > t$, since otherwise the trivial optimal solution is to include all the items. We assume $w_{\max} \le n^2$, because when $w_{\max} > n^2$ it is faster to run the textbook dynamic programming algorithm [Bel57] in $O(nt) \le O(n \cdot nw_{\max}) \le O(w_{\max}^2)$ time. We use the standard word-RAM computation model with $\Theta(\log n)$-bit words, and we assume $p_i \le p_{\max}$ fits into a single machine word.[5]

The *efficiency* of item i is p_i/w_i. We always assume the input items have *distinct* efficiencies p_i/w_i. This assumption is justified by the following tie-breaking lemma proved in Appendix B.1.

Lemma 2.1 (Break ties)**.** *Given a 0-1 Knapsack instance I, in $O(n)$ time we can deterministically reduce it to another 0-1 Knapsack instance I' with $n, w_{\max}$ and t unchanged, and $p'_{\max} \le \mathrm{poly}(p_{\max}, w_{\max}, n)$, such that the items in I' have* distinct efficiencies *and distinct profits.*

2.2 Greedy solution and proximity

Greedy solution. Sort the n input items in decreasing order of efficiency,

$$p_1/w_1 > p_2/w_2 > \cdots > p_n/w_n. \tag{3}$$

The *greedy solution* (or *maximal prefix solution*) is the item subset

$$G = \{1, 2, \ldots, i^*\}, \text{ where } i^* = \max\{i^* : w_1 + w_2 + \cdots + w_{i^*} \le t\}, \tag{4}$$

i.e., we greedily take the most efficient items one by one, until the next item cannot be added without exceeding the knapsack capacity. Since the input instance is nontrivial, we have $1 \le i^* \le n-1$, and $W(G) \in (t - w_{\max}, t]$. Denote the remaining items as $\overline{G} = [n] \setminus G = \{i^*+1, i^*+2, \ldots, n\}$.

[5] If this assumption is dropped, we simply pay an extra $O(\log p_{\max})$ factor in the running time for adding big integers.

Remark 2.2. As noted by [PRW21], the greedy solution G can be found in deterministic $O(n)$ time using linear-time median finding algorithms [BFP+73], as opposed to a straightforward $O(n \log n)$-time sorting according to Eq. (3).

Every item subset $X \subseteq [n]$ can be written as $X = (G \setminus B) \cup A$ where $A \subseteq \overline{G}$ and $B \subseteq G$. Finding an optimal knapsack solution X is equivalent to finding an *optimal exchange solution*, defined as a pair of subsets (A, B) $(A \subseteq \overline{G}, B \subseteq G)$ that maximizes $P(A) - P(B)$ subject to $W(A) - W(B) \leq t - W(G)$. Since any optimal knapsack solution X satisfies $W(X) \in (t - w_{\max}, t]$, we have

$$|W(A) - W(B)| = |W(G) - W(X)| < w_{\max} \tag{5}$$

for any optimal exchange solution (A, B).

Proximity. For any optimal exchange solution (A, B), a simple exchange argument shows that the weights of items in A and in B do not share any non-zero common subset sum, i.e.,

$$\mathcal{S}^*(\text{weights}(A)) \cap \mathcal{S}^*(\text{weights}(B)) = \varnothing. \tag{6}$$

Indeed, for an optimal knapsack solution $X = (G \setminus B) \cup A$, if non-empty item sets $A' \subseteq A$ and $B' \subseteq B$ have the same total weight, then $(X \cup B') \setminus A'$ is a set of items with the same total weight as X but *strictly* higher total profit (since efficiencies of items in $B' \subseteq G$ are strictly higher than efficiencies of items in $A' \subseteq \overline{G}$ due to Eqs. (3) and (4)), contradicting the optimality of X.

The following proximity bound Eq. (7) is consequence of Eq. (5) and Eq. (6), and was used in previous works such as [PRW21, CLMZ23] (see e.g., [PRW21, Lemma 2.1] for a short proof): for any optimal exchange solution (A, B), it holds that

$$|A| + |B| \leq 2w_{\max}. \tag{7}$$

In other words, any optimal knapsack solution X differs from the greedy solution G by at most $2w_{\max}$ items. This bound Eq. (7) immediately implies

$$W(A) + W(B) \leq 2w_{\max}^2 \tag{8}$$

for any optimal exchange solution (A, B).

Weight classes and ranks. We rank items of the same weight w according to their profits, as follows:

Definition 2.3 (Rank of items)**.** For each $w \in \mathcal{W}$, consider the weight-w items outside the greedy solution, $\overline{G} \cap I_{\{w\}} = \{i_1, i_2, \ldots, i_m\}$, where $p_{i_1} > p_{i_2} > \cdots > p_{i_m}$. We define $\text{rank}(i_1) = 1, \text{rank}(i_2) = 2, \ldots, \text{rank}(i_m) = m$. Similarly, consider the weight-w items in the greedy solution, $G \cap I_{\{w\}} = \{i'_1, i'_2, \ldots, i'_{m'}\}$, where $p_{i'_1} < p_{i'_2} < \cdots < p_{i'_{m'}}$. We define $\text{rank}(i'_1) = 1, \text{rank}(i'_2) = 2, \ldots, \text{rank}(i'_{m'}) = m'$. In this way, every item $i \in [n]$ receives a $\text{rank}(i)$.

Then, a standard observation is that an optimal solution should always take a prefix from each weight class:

Lemma 2.4 (Prefix property)**.** *Consider any optimal exchange solution (A, B). If $i \in A$, then $\{i' \in \overline{G} \cap I_{w_i} : \text{rank}(i') \leq \text{rank}(i)\} \subseteq A$, and $\text{rank}(i) \leq 2w_{\max}$.*

Similarly, if $i \in B$, then $\{i' \in G \cap I_{w_i} : \text{rank}(i') \leq \text{rank}(i)\} \subseteq B$, and $\text{rank}(i) \leq 2w_{\max}$.

Proof. We only prove the $i \in A$ case. The $i \in B$ case is symmetric. Consider two weight-w items $i, i' \in \overline{G} \cap I_w$ with $\mathrm{rank}(i') < \mathrm{rank}(i)$ and $i \in A$. Suppose for contradiction that $i' \notin A$. Then, $p_{i'} > p_i$, and hence $((A \setminus \{i\}) \cup \{i'\}, B)$ is an exchange solution with the same weight as (A, B) but achieving strictly higher profit, contradicting the optimality of (A, B). Hence, A contains all $i' \in \overline{G} \cap I_w$ with $\mathrm{rank}(i') \le \mathrm{rank}(i)$. Since $|A| \le 2w_{\max}$ by Eq. (7), we must have $\mathrm{rank}(i) \le 2w_{\max}$ for $i \in A$. □

We remark that all items $i \in [n]$ with $\mathrm{rank}(i) \le 2w_{\max}$ can be deterministically selected and sorted in $O(n + w_{\max}^2 \log w_{\max})$ time using linear-time median selection algorithms [BFP+73].

2.3 Dynamic programming and partial solutions

Our algorithm uses dynamic programming (DP) to find an optimal exchange solution (A, B) ($A \subseteq \overline{G}, B \subseteq G$). Now we introduce a few terminologies that will help us describe our DP algorithm later.

Definition 2.5 (Partial solutions and I-optimality)**.** A *partial exchange solution* (or simply a *partial solution*) refers to a pair of item subsets (A', B') where $A' \subseteq \overline{G}, B' \subseteq G$. The *weight* and *profit* of the partial solution (A', B') are defined as $W(A') - W(B')$ and $P(A') - P(B')$ respectively.

Let $I \subseteq [n]$ be an item subset. We say the partial solution (A', B') is *supported on* I if $A' \cup B' \subseteq I$. We say (A', B') is *I-optimal*, if there exits an optimal exchange solution (A, B) such that $A' = A \cap I$ and $B' = B \cap I$.

Definition 2.6 (DP tables)**.** A *DP table of size L* is an array $q[-L \,..\, L]$ with entries $q[z] \in \mathbb{Z} \cup \{-\infty\}$ for $z \in \{-L, \dots, L\}$.[6] (By convention, assume $q[z] = -\infty$ for $|z| > L$.) We omit its index range and simply write $q[\,]$ whenever its size is clear from context or is unimportant.

For an item subset $I \subseteq [n]$, we say $q[\,]$ is an *I-valid DP table*, if for every entry $q[z] \neq -\infty$ there exists a corresponding partial solution (A', B') supported on I with weight $W(A') - W(B') = z$ and profit $P(A') - P(B') = q[z]$. An I-valid DP table $q[\,]$ is said to be *I-optimal* if it contains some entry $q[z]$ that corresponds to an I-optimal partial solution.

For example, the trivial DP table with $q[0] = 0, q[z] = -\infty (z \neq 0)$ is $\varnothing$-optimal (it contains the empty partial solution $(\varnothing, \varnothing)$). In dynamic programming we gradually extend this $\varnothing$-optimal DP table to an $[n]$-optimal DP table which should contain an optimal exchange solution. As a basic example, given an I-optimal DP table $q[-L \,..\, L]$, for $i \notin I$ we can obtain an $(I \cup \{i\})$-optimal DP table $q'[-L - w_i \,..\, L + w_i]$ in $O(L + w_i)$ time via the update rule $q'[z] \leftarrow \max\{q[z], q[z \mp w_i] \pm p_i\}$ (where $\pm$ is $+$ if $i \in \overline{G}$, or $-$ if $i \in G$).

Previous dynamic programming algorithms for 0-1 Knapsack [PRW21, CLMZ23, AT19, KP04] used the following standard lemma based on the SMAWK algorithm [AKM+87]:

Lemma 2.7 (Batch-updating items of the same weight)**.** *Let $I \subseteq [n]$ and $J \subseteq \overline{G}$ be disjoint item subsets, and all $j \in J$ have the same weight $w_j = w$. Suppose an upper bound L' is known such that all $(I \cup J)$-optimal partial solutions (A', B') satisfy $|W(A') - W(B')| \le L'$.*

Then, given an I-optimal DP table $q[-L \,..\, L]$, we can compute an $(I \cup J)$-optimal DP table $q'[-L' \,..\, L']$ in $O(L + L' + |J| + w_{\max} \log w_{\max})$ time.

The same statement holds if the assumption $J \subseteq \overline{G}$ is replaced by $J \subseteq G$.

[6] For brevity we call it size-L despite its actual length being $(2L + 1)$.

Proof Sketch. Let $J = \{j_1, j_2, \ldots, j_{|J|}\} \subseteq \overline{G}$ be sorted so that $p_{j_1} > p_{j_2} > \cdots > p_{j_{|J|}}$. By the prefix property (Lemma 2.4), for function $Q(x) := p_{j_1} + p_{j_2} + \cdots + p_{j_x}$ we know that $q'[z] := \max_{x \geq 0}\{q[z - xw] + Q(x)\}$ gives an $(I \cup J)$-optimal DP table $q'[-L' \,..\, L']$. Moreover, due to the size of the tables, it suffices to maximize over $x \geq 0$ where $xw \leq L + L'$, i.e., $0 \leq x \leq \lfloor (L + L')/w \rfloor$.

The task of computing $q'[\,]$ can be decomposed into w independent subproblems based on the remainder $r = (z \bmod w)$. Each subproblem then becomes computing the $(\max, +)$-convolution of the subsequence of q with indices $\equiv r \pmod w$ and the array $[Q(0), \ldots, Q(\lfloor (L + L')/w \rfloor)]$, which can be done in linear time using SMAWK algorithm [AKM+87] due to the concavity of $Q(\cdot)$ (i.e., $Q(x) - Q(x-1) \geq Q(x+1) - Q(x)$ for all $x \geq 1$). (Readers unfamiliar with the result of [AKM+87] may refer to Appendix A for an overview.) Hence, the total time for running SMAWK is linear in the total size of these arrays, which is $O(L + L' + w)$.

Finally we remark that we do not need to sort all items of J in $O(|J| \log |J|)$ time at the beginning. By Lemma 2.4, we only need to consider those with rank at most $2w_{\max}$, which can be selected from J and sorted in $O(|J| + w_{\max} \log w_{\max})$ time [BFP+73]. □

3 Algorithm for 0-1 Knapsack

In this section we present our algorithm for 0-1 Knapsack (Theorem 1.1). In Section 3.1, we recall a crucial weight partitioning lemma from [CLMZ23] based on fine-grained proximity, which naturally gives rise to a two-stage algorithm framework. The second stage can be easily performed using previous techniques [PRW21, CLMZ23] and is described in Section 3.1, while the first stage contains our main technical challenge and is described in Sections 3.2, 3.3 and 4: In Section 3.2, we give a rank partitioning lemma based on another proximity result. Given this lemma, in Section 3.3 we abstract out a core subproblem called HintedKnapsackExtend$^+$, and describe how to implement the first stage of our algorithm assuming this core subproblem can be solved efficiently. Our algorithm for HintedKnapsackExtend$^+$ will be described in Section 4.

3.1 Weight partitioning and the second-stage algorithm

Chen, Lian, Mao, and Zhang [CLMZ23] recently used additive-combinatorial results of Bringmann and Wellnitz [BW21] to obtain several powerful structural lemmas involving the support size of two integer multisets A, B avoiding non-zero common subset sums. These structural results (called "fine-grained proximity" in [CLMZ23]) allowed them to obtain faster knapsack algorithms than the earlier works [PRW21, EW20] based on ℓ_1-proximity (Eq. (7)) only. Here we recall one of the key lemmas from [CLMZ23].[7]

Lemma 3.1 ([CLMZ23, Lemma 11], paraphrased)**.** *There is a constant C such that the following holds. Suppose two multisets A, B supported on $[N]$ satisfy*

$$|\operatorname{supp}(A)| \geq C\sqrt{N \log N}$$

and

$$\Sigma(B) \geq \frac{CN^2\sqrt{\log N}}{|\operatorname{supp}(A)|}.$$

Then, $\mathcal{S}^*(A) \cap \mathcal{S}^*(B) \neq \varnothing$.

[7] The original statement of [CLMZ23, Lemma 11] had a worse $\log N$ factor than the $\sqrt{\log N}$ factor in Lemma 3.1. By inspection of their proof, they actually proved the stronger version stated here in Lemma 3.1.

Using this fine-grained proximity result, Chen, Lian, Mao, and Zhang obtained a weight partitioning lemma [CLMZ23, Lemma 13], which is a key ingredient in their algorithm. Our algorithm also crucially relies on this weight partitioning lemma in a similar way, but for our purpose we need to extend it from the two-partition version in [CLMZ23] to $O(\log w_{\max})$-partition.[8]

Recall the following notations from Section 2.1: $W(I) = \sum_{i \in I} w_i$, $\mathcal{W} = \mathrm{supp}(\{w_1, w_2, \ldots, w_n\}) \subseteq [w_{\max}]$, and $I_{\mathcal{W}'} := \{i \in [n] : w_i \in \mathcal{W}'\}$.

Lemma 3.2 (Extension of [CLMZ23, Lemma 13])**.** *The set $\mathcal{W}$ of input item weights can be partitioned in $O(n + w_{\max} \log w_{\max})$ time into $\mathcal{W} = \mathcal{W}_1 \uplus \mathcal{W}_2 \uplus \cdots \uplus \mathcal{W}_s$, where $s < \log_2(\sqrt{w_{\max}})$, with the following property:*

Denote $\mathcal{W}_{\le j} = \mathcal{W}_1 \cup \cdots \cup \mathcal{W}_j$ and $\mathcal{W}_{>j} = \mathcal{W} \setminus \mathcal{W}_{\le j}$. For every optimal exchange solution (A, B) and every $1 \le j \le s$,

- $|\mathcal{W}_j| \le 4C\sqrt{w_{\max} \log w_{\max}} \cdot 2^j$, *and*
- $W(A \cap I_{\mathcal{W}_{>j}}) \le 4C w_{\max}^{3/2}/2^j$ *and* $W(B \cap I_{\mathcal{W}_{>j}}) \le 4C w_{\max}^{3/2}/2^j$,

where C is the universal constant from Lemma 3.1.

The proof of Lemma 3.2 is similar to that of the original two-partition version [CLMZ23, Lemma 13], and is deferred to Appendix B.2.

Given this weight partitioning $\mathcal{W} = \mathcal{W}_1 \uplus \mathcal{W}_2 \uplus \cdots \uplus \mathcal{W}_s$, our overall algorithm runs in two stages: in the first stage, we only consider items whose weights belong to $\mathcal{W}_1$, and efficiently compute an $I_{\mathcal{W}_1}$-optimal DP table (see Definition 2.6) by exploiting the small size of $\mathcal{W}_1$. Then, the second stage of the algorithm updates the DP table using the remaining items $I_{\mathcal{W}_2} \uplus \cdots \uplus I_{\mathcal{W}_s}$. The second stage follows the same idea as [CLMZ23] of using Lemma 3.2 to trade off the size of the DP table and the number of linear-time scans needed to update the DP table. In contrast, the first stage is more technically challenging; we summarize it in the following lemma, and prove it in subsequent sections:

Lemma 3.3 (The first stage)**.** *Let $\mathcal{W}_1 \subseteq [w_{\max}]$ from Lemma 3.2 be given. Then we can compute an $I_{\mathcal{W}_1}$-optimal DP table in $O(n + w_{\max}^2 \log^4 w_{\max})$ time.*

The overall $O(n + w_{\max}^2 \log^4 w_{\max})$ algorithm for 0-1 Knapsack then follows from Lemma 3.3 and Lemma 3.2, using arguments similar to [CLMZ23].

Proof of Theorem 1.1 assuming Lemma 3.3. Use Lemma 3.2 to obtain the weight partition $\mathcal{W} = \mathcal{W}_1 \uplus \mathcal{W}_2 \uplus \cdots \uplus \mathcal{W}_s$, where $s < \log_2(\sqrt{w_{\max}})$. Define a size upper bound $L_j := 4C w_{\max}^{3/2}/2^j + w_{\max}$ for $1 \le j \le s$. Note that for every optimal exchange solution (A, B) and every $1 \le j \le s$ we have

$$\begin{aligned} |W(A \cap I_{\mathcal{W}_{\le j}}) - W(B \cap I_{\mathcal{W}_{\le j}})| &= \big|\big(W(A) - W(B)\big) - \big(W(A \cap I_{\mathcal{W}_{>j}}) - W(B \cap I_{\mathcal{W}_{>j}})\big)\big| \\ &\le |W(A) - W(B)| + |W(A \cap I_{\mathcal{W}_{>j}}) - W(B \cap I_{\mathcal{W}_{>j}})| \\ &< w_{\max} + 4C w_{\max}^{3/2}/2^j \\ &= L_j, \end{aligned} \tag{9}$$

where the last inequality follows from Eq. (5) and Lemma 3.2.

[8] We remark that [CLMZ23, Lemma 18] also gave a three-partition extension of this lemma, but in a different way than what we need here.

We run the first-stage algorithm of Lemma 3.3, and obtain an $I_{\mathcal{W}_1}$-optimal DP table. By Eq. (9), we can shrink the size of this DP table to L_1 without losing its $I_{\mathcal{W}_1}$-optimality. Then, we repeatedly apply the following claim:

Claim 3.4. *For $2 \le j \le s$, given an $I_{\mathcal{W}_{\le j-1}}$-optimal DP table of size L_{j-1}, we can compute an $I_{\mathcal{W}_{\le j}}$-optimal DP table of size L_j in $O(|\mathcal{W}_j|(L_{j-1} + w_{\max}\log w_{\max}) + |I_{\mathcal{W}_j}|)$ time.*

Proof of Claim 3.4. We need to update the DP table using items from $I_{\mathcal{W}_j}$. Partition them into $I^+_{\mathcal{W}_j} = I_{\mathcal{W}_j} \cap \overline{G}$ and $I^-_{\mathcal{W}_j} = I_{\mathcal{W}_j} \cap G$ (recall from Section 2.2 that G denotes the greedy solution). We first update the DP table with the "positive items" $I^+_{\mathcal{W}_j}$. By a similar proof to Eq. (9), we know

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

holds for any optimal exchange solution (A, B). Hence, we can iterate over $w \in \mathcal{W}_j$, and use the batch-update lemma based on SMAWK (Lemma 2.7) to update the size-L_{j-1} DP table with the weight-w items $I_{\{w\}} \cap \overline{G}$, in $O(L_{j-1} + |I_{\{w\}} \cap \overline{G}| + w_{\max}\log w_{\max})$ time. In the end we obtain a $(I_{\mathcal{W}_{\le j-1}} \cup I^+_{\mathcal{W}_j})$-optimal DP table of size L_{j-1}, in total time $\sum_{w\in\mathcal{W}_j} O(L_{j-1} + |I_{\{w\}} \cap \overline{G}| + w_{\max}\log w_{\max}) = O(|\mathcal{W}_j|(L_{j-1} + w_{\max}\log w_{\max}) + |I_{\mathcal{W}_j} \cap \overline{G}|)$.

Then, we update the $(I_{\mathcal{W}_{\le j-1}} \cup I^+_{\mathcal{W}_j})$-optimal DP table with the "negative items" $I^-_{\mathcal{W}_j}$, and obtain an $I_{\mathcal{W}_{\le j}}$-optimal DP table. This algorithm is symmetric to the positive case described above, and similarly runs in time $O(|\mathcal{W}_j|(L_{j-1} + w_{\max}\log w_{\max}) + |I_{\mathcal{W}_j} \cap G|)$.

In the end, we shrink the size of the obtained DP table to L_j. By Eq. (9), this does not affect the $I_{\mathcal{W}_{\le j}}$-optimality of the DP table. The total running time is $O(|\mathcal{W}_j|(L_{j-1} + w_{\max}\log w_{\max}) + |I_{\mathcal{W}_j}|)$. □

In the main algorithm, starting from the $I_{\mathcal{W}_1}$-optimal DP table, we sequentially apply Claim 3.4 for $j = 2, 3, \ldots, s$, and in the end we can obtain an $I_{\mathcal{W}_{\le s}}$-optimal DP table. Note that $I_{\mathcal{W}_{\le s}} = [n]$, so the final DP table contains an optimal exchange solution. The total time for applying Claim 3.4 is (up to a constant factor)

$$\begin{aligned}
&\sum_{j=2}^{s} \left(|\mathcal{W}_j|(L_{j-1} + w_{\max}\log w_{\max}) + |I_{\mathcal{W}_j}|\right) \\
\le\ &\sum_{j=2}^{s} 4C\sqrt{w_{\max}\log w_{\max}} \cdot 2^j \cdot (4Cw_{\max}^{3/2}/2^{j-1} + 2w_{\max}\log w_{\max}) + \sum_{j=2}^{s} |I_{\mathcal{W}_j}| \\
=\ &\sum_{j=2}^{s} \left(32C^2 w_{\max}^2\sqrt{\log w_{\max}} + 8C \cdot 2^j \cdot (w_{\max}\log w_{\max})^{3/2}\right) + \sum_{j=2}^{s} |I_{\mathcal{W}_j}| \\
\le\ &O(w_{\max}^2(\log w_{\max})^{3/2} + n). \qquad \text{(by } s < \log_2(\sqrt{w_{\max}})\text{)}
\end{aligned}$$

The time complexity of the entire algorithm is dominated by the $O(n + w_{\max}^2 \log^4 w_{\max})$ running time of the first stage (Lemma 3.3). □

3.2 Rank partitioning

Given $\mathcal{W}_1 \subseteq [w_{\max}]$ of size $|\mathcal{W}_1| \le O(\sqrt{w_{\max}\log w_{\max}})$ from Lemma 3.2, we partition the items whose weights belong to $\mathcal{W}_1$ into dyadic groups based on their ranks (Definition 2.3), as follows:

Definition 3.5 (Rank partitioning)**.** Let $k = \lceil \log_2(2w_{\max} + 1) \rceil$. For each $1 \le j \le k$, define item subsets

$$J_j^+ := \{i \in \overline{G} \cap I_{\mathcal{W}_1} : 2^{j-1} \le \mathrm{rank}(i) \le 2^j - 1\},$$

and

$$J_j^- := \{i \in G \cap I_{\mathcal{W}_1} : 2^{j-1} \le \mathrm{rank}(i) \le 2^j - 1\}.$$

Note that $J_1^+ \uplus J_1^- \uplus \cdots \uplus J_k^+ \uplus J_k^-$ form a partition of $\{i \in I_{\mathcal{W}_1} : \mathrm{rank}(i) \le 2^k - 1\}$.

Denote

$$J_{\le j}^+ = J_1^+ \cup \cdots \cup J_j^+,$$

and

$$J_{\le j}^- = J_1^- \cup \cdots \cup J_j^-.$$

Note the the rank partitioning defined in Definition 3.5 can be computed in $O(n + w_{\max} \log w_{\max})$ time.

Our rank partitioning is motivated by the following additive-combinatorial Lemma 3.6, which can be derived from the results of Bringmann and Wellnitz [BW21]. Recall the *r-support* $\mathrm{supp}_r(X)$ of a multiset X is the set of items in X with multiplicity at least r.

Lemma 3.6. *There is a constant C such that the following holds. Suppose two multisets A, B supported on $[N]$ satisfy*

$$|\mathrm{supp}_r(A)| \ge C\sqrt{N/r} \cdot \sqrt{\log(2N)} \tag{10}$$

for some $r \ge 1$, and

$$\Sigma(B) \ge \Sigma(A) - N. \tag{11}$$

Then, $\mathcal{S}^*(A) \cap \mathcal{S}^*(B) \ne \varnothing$.

Lemma 3.6 is partly inspired by [CLMZ23, Lemma 12] which generalized their fine-grained proximity result (Lemma 3.1) from $\mathrm{supp}(A)$ to $\mathrm{supp}_r(A)$.[9] We include a proof of Lemma 3.6 in Appendix B.3.

Using Lemma 3.6, we obtain the following structural lemma for the rank partitioning. Recall the definition of $I_{\mathcal{W}_1}$-optimal partial solutions from Definition 2.5.

Lemma 3.7 (Rank partitioning structural lemma)**.** *For a universal constant C, the partition $J_1^+ \uplus J_1^- \uplus \cdots \uplus J_k^+ \uplus J_k^- \subseteq I_{\mathcal{W}_1}$ from Definition 3.5 satisfies the following properties for every $I_{\mathcal{W}_1}$-optimal partial solution (A', B'):*

1. *$A' \subseteq J_{\le k}^+$ and $B' \subseteq J_{\le k}^-$.*
2. *For all $1 \le j \le k$, $|A' \cap J_{\le j}^+| \le m_j$ and $|B' \cap J_{\le j}^-| \le m_j$, where*

$$m_j := C \cdot 2^{j/2} \cdot \sqrt{w_{\max} \log(2w_{\max})}. \tag{12}$$

[9] Their generalization of Lemma 3.1 is not applicable in our first-stage algorithm. Note that Lemma 3.6 is incomparable to Lemma 3.1 even when $r = 1$.

3. *For all* $1 \le j \le k$,

$$|\{w \in \mathcal{W}_1 : I_{\{w\}} \cap J^+_{j-1} \subseteq A' \text{ and } I_{\{w\}} \cap J^+_j \neq \varnothing\}| \le b_j,$$

and similarly

$$|\{w \in \mathcal{W}_1 : I_{\{w\}} \cap J^-_{j-1} \subseteq B' \text{ and } I_{\{w\}} \cap J^-_j \neq \varnothing\}| \le b_j,$$

where

$$b_j := C \cdot 2^{-j/2} \cdot \sqrt{w_{\max} \log(2w_{\max})}, \tag{13}$$

and $J^+_0 := J^-_0 := \varnothing$.

Proof. Let (A, B) be an optimal exchange solution such that $A \cap I_{\mathcal{W}_1} = A'$ and $B \cap I_{\mathcal{W}_1} = B'$. In the following we will only prove the claimed properties about A', and the properties about B' can be proved similarly.

For Item 1, note that $J^+_{\le k} = \{i \in \overline{G} \cap I_{\mathcal{W}_1} : \operatorname{rank}(i) \le 2^k - 1\}$, and we have $2^k - 1 \ge 2w_{\max}$ by the definition of k. By the prefix property (Lemma 2.4), we have $\operatorname{rank}(i) \le 2w_{\max}$ for all $i \in A'$, and thus $A' \subseteq J^+_{\le k}$.

For Item 2, we apply Lemma 3.6 with $N := w_{\max}$ to the multisets $\text{weights}(A)$ and $\text{weights}(B)$, which should not share any common non-zero subset sum (Eq. (6)). Since $|W(A) - W(B)| < w_{\max}$ (Eq. (5)), Lemma 3.6 implies for all $r \ge 1$ that

$$|\operatorname{supp}_r(\text{weights}(A))| < C_0 \sqrt{w_{\max}/r} \cdot \sqrt{\log(2w_{\max})} \tag{14}$$

for some universal constant C_0. Hence,

$$\begin{aligned}
|A' \cap J^+_{\le j}| &= |\{i \in A' : 1 \le \operatorname{rank}(i) \le 2^j - 1\}| \\
&= \sum_{w \in \mathcal{W}_1} \min\{2^j - 1, |A' \cap I_{\{w\}}|\} \\
&= \sum_{r=1}^{2^j-1} |\operatorname{supp}_r(\text{weights}(A'))| \\
&\le \sum_{r=1}^{2^j-1} |\operatorname{supp}_r(\text{weights}(A))| \\
&\le \sum_{r=1}^{2^j-1} C_0 \sqrt{w_{\max}/r} \cdot \sqrt{\log(2w_{\max})} && \text{(by Eq. (14))} \\
&< C_0 \sqrt{w_{\max}} \cdot 2\sqrt{2^j - 1} \cdot \sqrt{\log(2w_{\max})} \\
&< C 2^{j/2} \sqrt{w_{\max} \log(2w_{\max})}
\end{aligned}$$

for some universal constant C.

For Item 3, the set under consideration is a subset of $\mathcal{W}_1$, and in the $j = 1$ case we can simply bound its size using Lemma 3.2 as $\le |\mathcal{W}_1| \le 8C_1 \sqrt{w_{\max} \log w_{\max}} \le b_1$, provided the constant C in the definition of b_1 (Eq. (13)) is large enough. Now it remains to consider $2 \le j \le k$, and we have to bound the number of $w \in \mathcal{W}_1$ such that $I_{\{w\}} \cap J^+_{j-1} \subseteq A'$ and $I_{\{w\}} \cap J^+_j \neq \varnothing$. For any such w, by definition of our rank partitioning (Definition 3.5), $I_{\{w\}} \cap J^+_j \neq \varnothing$ means that $I_{\{w\}} \cap \overline{G}$ contains

some item i with $\mathrm{rank}(i) \geq 2^{j-1}$, and hence $|I_{\{w\}} \cap J^+_{j-1}| = 2^{j-2}$. Then from $I_{\{w\}} \cap J^+_{j-1} \subseteq A'$ we get $|I_{\{w\}} \cap A'| \geq 2^{j-2}$, or equivalently, $w \in \mathrm{supp}_{2^{j-2}}(\mathrm{weights}(A'))$. Thus, the number of such w is at most

$$\begin{aligned}
|\mathrm{supp}_{2^{j-2}}(\mathrm{weights}(A'))| &\leq |\mathrm{supp}_{2^{j-2}}(\mathrm{weights}(A))| \\
&< C_0\sqrt{w_{\max}/(2^{j-2})} \cdot \sqrt{\log(2w_{\max})} && \text{(by Eq. (14))} \\
&\leq C\sqrt{w_{\max}} \cdot 2^{-j/2}\sqrt{\log(2w_{\max})} \\
&= b_j
\end{aligned}$$

for some large enough constant C. □

3.3 The first-stage algorithm via hinted dynamic programming

Based on our rank partitioning $J^+_1 \uplus J^-_1 \uplus \cdots \uplus J^+_k \uplus J^-_k \subseteq I_{\mathcal{W}_1}$, $k = \lceil \log_2(2w_{\max}+1) \rceil$ (Definition 3.5) and its structural lemma (Lemma 3.7), our first-stage algorithm uses dynamic programming and runs in k phases. At the beginning of the j-th phase ($1 \leq j \leq k$), we have a $(J^+_{\leq j-1} \cup J^-_{\leq j-1})$-optimal DP table, and we first update it with the "positive items" J^+_j to obtain a $(J^+_{\leq j} \cup J^-_{\leq j-1})$-optimal DP table, and then update it with the "negative items" J^-_j to obtain a $(J^+_{\leq j} \cup J^-_{\leq j})$-optimal DP table. We will adjust the size of the DP table throughout the k phases based on Item 2 of Lemma 3.7. This is similar to the second-stage algorithm from Section 3.1, except that in Section 3.1 the DP table is shrinking whereas here it will be expanding.

To implement the DP efficiently, we crucially rely on Item 3 of Lemma 3.7, which gives an upper bound on the "active support" of the weights of items in every partial solution in the current DP table. More specifically, consider an $I_{\mathcal{W}_1}$-optimal partial solution (A', B') and its restriction (A'', B'') where $A'' = A' \cap J^+_{\leq j-1}, B'' = B' \cap J^-_{\leq j-1}$. Then Item 3 of Lemma 3.7 implies that the items in $A' \setminus A''$ (or $B' \setminus B''$) can only have at most b_j distinct weights. This means that, for any partial solution (A'', B'') in the DP table at the end of phase $j-1$, in order to extend it to an $I_{\mathcal{W}_1}$-optimal partial solution (A', B') in future phases, we only need to update it with items from these b_j weight classes determined by Item 3 of Lemma 3.7. This idea is called *witness propagation*, and was originally introduced by Deng, Mao, and Zhong [DMZ23] in the context of unbounded knapsack-type problems. Implementing this idea in the more difficult 0-1 setting is a main technical contribution of this paper.

In the rest of this section, we will introduce a few more definitions to help use formally describe our algorithm, and we will abstract out a core subproblem called HintedKnapsackExtend$^+$ which captures the aforementioned idea of witness propagation. Then we will show how to implement our first-stage algorithm and prove Lemma 3.3, assuming HintedKnapsackExtend$^+$ can be solved efficiently.

In the following definition, we augment each entry of the DP table with hints, which contain the weight classes from which we need to add items when we update this entry, as we just discussed.

Definition 3.8 (Hinted DP tables)**.** A *hinted DP table* is a DP table $q[\,]$ where each entry $q[z] \neq -\infty$ is annotated with two sets $S^+[z], S^-[z] \subseteq \mathcal{W}_1$. We say the table has *positive hint size* b if $|S^+[z]| \leq b$ for all z, and has *negative hint size* b if $|S^-[z]| \leq b$ for all z.

For an item subset $J \subseteq I_{\mathcal{W}_1}$, we say a hinted DP table $q[\,]$ is *hinted-J-optimal*, if $q[\,]$ is J-valid (see Definition 2.6), and it has an entry $q[z]$ such that both of the following hold:

1. There exists an $I_{\mathcal{W}_1}$-optimal partial solution (A', B') such that $W(A' \cap J) - W(B' \cap J) = z$ and $P(A' \cap J) - P(B' \cap J) = q[z]$.

2. Every $I_{\mathcal{W}_1}$-optimal partial solution (A', B') with $W(A' \cap J) - W(B' \cap J) = z$ should satisfy $A' \setminus J \subseteq I_{S^+[z]}$ and $B' \setminus J \subseteq I_{S^-[z]}$.

Note that if a hinted DP table is hinted-J-optimal, then in particular it is J-optimal in the sense of Definition 2.6 (due to Item 1 of Definition 3.8).

The following lemma summarizes each of the $k = \lceil \log_2(2w_{\max} + 1) \rceil$ phases in our first-stage algorithm.

Lemma 3.9. *Let k, m_j, b_j be defined as in Lemma 3.7. Let $L_j := m_j \cdot w_{\max}$. For every $1 \le j \le k$, the following hold:*

1. *Given a hinted-$(J^+_{\le j-1} \cup J^-_{\le j-1})$-optimal DP table of size L_{j-1} with positive and negative hint size b_j, we can compute a hinted-$(J^+_{\le j} \cup J^-_{\le j-1})$-optimal DP table of size L_j with positive hint size b_{j+1} and negative hint size b_j, in $O(L_j b_j \cdot \log^2(L_j b_j) + |J^+_j|)$ time.*

2. *Given a hinted-$(J^+_{\le j} \cup J^-_{\le j-1})$-optimal DP table of size L_j with positive hint size b_{j+1} and negative hint size b_j, we can compute a hinted-$(J^+_{\le j} \cup J^-_{\le j})$-optimal DP table of size L_j with positive and negative hint size b_{j+1}, in $O(L_j b_j \cdot \log^2(L_j b_j) + |J^-_j|)$ time.*

Lemma 3.9 immediately implies our overall first-stage algorithm:

Proof of Lemma 3.3 assuming Lemma 3.9. We start with the trivial hinted DP table with $q[0] = 0$ and $S^+[0] = S^-[0] = \mathcal{W}_1$ (padded with $q[z] = -\infty$ for $z \in \{-L_0, \dots, L_0\} \setminus \{0\}$). By definition, $q[\,]$ is hinted-$\varnothing$-optimal, and has positive and negative hint size $|\mathcal{W}_1| \le b_1$. Then, we iteratively perform phases $j = 1, 2, \dots, k$, where in phase j we first apply Item 1 of Lemma 3.9, and then apply Item 2 of Lemma 3.9. In the end of phase k, we have obtained a hinted-$(J^+_{\le k} \cup J^-_{\le k})$-optimal DP table. By Item 1 of Lemma 3.7, it is an $I_{\mathcal{W}_1}$-optimal DP table as desired.

The total time of applying Lemma 3.9 is (up to a constant factor)

$$
\begin{aligned}
&\sum_{j=1}^{k} \left(L_j b_j \cdot \log^2(L_j b_j) + |J^+_j| + |J^-_j| \right) \\
\le\ &\sum_{j=1}^{k} m_j w_{\max} b_j \log^2(m_j w_{\max} b_j) \ + n \\
=\ &\sum_{j=1}^{k} C^2 w^2_{\max} \log(2w_{\max}) \log^2 \left(C^2 w^2_{\max} \log(2w_{\max}) \right) \ + n && \text{(by Eqs. (12) and (13))} \\
\le\ &O(w^2_{\max} \log^4 w_{\max} + n). && \text{(by } k = \lceil \log_2(2w_{\max} + 1) \rceil\text{)}
\end{aligned}
$$

□

It remains to prove Lemma 3.9. In the following, we will reduce it to a core subproblem called HintedKnapsackExtend$^+$, which captures the task of updating a hinted size-L DP table with positive hint size b using "positive items" whose weights come from some positive integer set U (here

we can think of $U = \mathcal{W}_1$). Similarly to the proof of the batch-update lemma (Lemma 2.7) based on SMAWK, here we also use a function $Q_w \colon \mathbb{Z}_{\geq 0} \to \mathbb{Z}$ to represent the total profit of taking the top-x items of weight w.

Problem 1 (HintedKnapsackExtend$^+$)**.** *Let $U \subseteq \mathcal{W}_1$. For every $w \in U$, suppose $Q_w \colon \mathbb{Z}_{\geq 0} \to \mathbb{Z}$ is a concave function with $Q_w(0) = 0$ that can be evaluated in constant time. We are given a DP table $q[-L \mathinner{..} L]$ (where $q[i] \in \mathbb{Z} \cup \{-\infty\}$), annotated with $S[-L \mathinner{..} L]$ where $S[i] \subseteq U$.*

Consider the following optimization problem for each $-L \leq i \leq L$: find a solution vector $\boldsymbol{x}[i] \in \mathbb{Z}_{\geq 0}^U$ that maximizes the total profit

$$r[i] := q\big[z[i]\big] + \sum_{w \in U} Q_w(x[i]_w), \tag{15}$$

where $z[i] \in \mathbb{Z}$ is uniquely determined by

$$z[i] + \sum_{w \in U} w \cdot x[i]_w = i. \tag{16}$$

The task is to solve this optimization problem for each $-L \leq i \leq L$ with the following relaxation*:*

- *If* all *maximizers $(\boldsymbol{x}[i], z[i])$ of Eq. (15) (subject to Eq. (16)) satisfy*

$$\operatorname{supp}(\boldsymbol{x}[i]) \subseteq S\big[z[i]\big], \tag{17}$$

  then we are required to correctly output a maximizer for i.

- *Otherwise, we are allowed to output a suboptimal solution for i.*

Remark 3.10. We give a few remarks to help get a better understanding of Problem 1:

1. In Eq. (16), $z[i] \leq i$ must hold, since $w \in U \subseteq [w_{\max}]$ is always positive and $\boldsymbol{x}[i]$ is a non-negative vector.

2. If we do not have the relaxation based on hints $S[i]$, then Problem 1 becomes a standard problem solvable in $O(|U|L)$ time using SMAWK algorithm (basically, repeat the proof of Lemma 2.7 for every $w \in U$; see also [PRW21, CLMZ23, AT19, KP04]).

3. Under this relaxation, without loss of generality, we can assume the output of Problem 1 always satisfies $\operatorname{supp}(\boldsymbol{x}[i]) \subseteq S\big[z[i]\big]$ (Eq. (17)) for all $-L \leq i \leq L$. (If we had to output an $\boldsymbol{x}[i]$ that violates Eq. (17), then we must be in the "otherwise" case for i, and should be allowed to output anything). In particular, if $|S[i]| \leq b$ for all $-L \leq i \leq L$, then we can assume the output $\boldsymbol{x}[-L \mathinner{..} L]$ of Problem 1 has description size $O(bL)$ words.

4. Note that Problem 1 is different from (and easier than) the task of maximizing Eq. (15) for every i subject to Eq. (17). The latter version would make a cleaner definition, but it is a harder problem which we do not know how to solve.

The following Theorem 3.11 summarizes our algorithm for Problem 1, which will be given in Section 4.

Theorem 3.11. *HintedKnapsackExtend$^+$ (Problem 1) with $|S[i]| \leq b$ for all $-L \leq i \leq L$ can be solved deterministically in $O(Lb \log^2(Lb))$ time.*

Finally, we show how to prove Lemma 3.9 using Theorem 3.11.

Proof of Lemma 3.9 assuming Theorem 3.11. Here we only prove Item 1 of Lemma 3.9 (Item 2 can be proved similarly in the reverse direction). Given a hinted-$(J^+_{\le j-1} \cup J^-_{\le j-1})$-optimal DP table $q[-L_{j-1}\,..\,L_{j-1}]$ with positive hint size $|S^+[i]| \le b_j$ and negative hint size $|S^-[i]| \le b_j$, our task is to compute a hinted-$(J^+_{\le j} \cup J^-_{\le j-1})$-optimal DP table $r[\,]$, by updating $q[\,]$ with items from J^+_j.

As usual, partition J^+_j into weight classes: for each $w \in \mathcal{W}_1$, let the weight-w items $J^+_j \cap I_{\{w\}} = \{i_1, i_2, \ldots, i_m\}$ be sorted so that $p_{i_1} > p_{i_2} > \cdots > p_{i_m}$, and define their top-$x$ total profit $Q_w(x) := p_{i_1} + p_{i_2} + \cdots + p_{i_x}$, which is a concave function in x. After preprocessing in $O(|J^+_j| + |\mathcal{W}_1| w_{\max} \log w_{\max})$ total time, $Q_w(\cdot)$ can be evaluated in constant time for all $w \in \mathcal{W}_1$.

Then, we enlarge the DP table $q[-L_{j-1}\,..\,L_{j-1}]$ to $q[-L_j\,..\,L_j]$ by padding dummy entries $-\infty$, and define a HintedKnapsackExtend$^+$ instance on functions Q_w and DP table $q[-L_j\,..\,L_j]$ with hints $S[-L_j\,..\,L_j] := S^+[-L_j\,..\,L_j]$ (recall $S^+[i] \subseteq \mathcal{W}_1$ and $|S^+[i]| \le b_j$). We run the algorithm from Theorem 3.11 to solve this instance in $O(L_j b_j \log^2(L_j b_j))$ time, and obtain solution vectors $\boldsymbol{x}[i] \in \mathbb{Z}^{\mathcal{W}_1}_{\ge 0}$ for all $-L_j \le i \le L_j$. Recall from Eqs. (15) and (16) that solution vector $\boldsymbol{x}[i]$ achieves total profit

$$r[i] := q\big[z[i]\big] + \sum_{w \in \mathcal{W}_1} Q_w(x[i]_w),$$

where $z[i] + \sum_{w \in \mathcal{W}_1} w \cdot x[i]_w = i$.

In the following, we will first show that $r[-L_j\,..\,L_j]$ satisfies Item 1 of Definition 3.8 (and thus is an $(J^+_{\le j} \cup J^-_{\le j-1})$-optimal DP table). Then we will compute new hints $T^+[-L_j\,..\,L_j], T^-[-L_j\,..\,L_j]$ to satisfy Item 2 of Definition 3.8, making $r[-L_j\,..\,L_j]$ a hinted-$(J^+_{\le j} \cup J^-_{\le j-1})$-optimal DP table.

Optimality. Since $q[\,]$ is a hinted-$(J^+_{\le j-1} \cup J^-_{\le j-1})$-optimal DP table by assumption, by Definition 3.8 (Item 1) there exists an $I_{\mathcal{W}_1}$-optimal partial solution (A', B') such that $q[\,]$ contains partial solution (A'', B'') where $A'' := A' \cap J^+_{\le j-1}, B'' := B' \cap J^-_{\le j-1}$, that is, we have $q[z''] = P(A'') - P(B'')$ for $z'' := W(A'') - W(B'')$. Now consider the partial solution $(\widehat{A}'', B'')$ where $\widehat{A}'' := A' \cap J^+_{\le j}$, and denote their difference by $Y := \widehat{A}'' \setminus A'' = A' \cap J^+_j$. Encode the items in Y by a vector $\boldsymbol{y} \in \mathbb{Z}^{\mathcal{W}_1}_{\ge 0}$ where $y_w = |I_{\{w\}} \cap Y|$, and by the prefix property (Lemma 2.4) we must have $P(Y) = \sum_{w \in \mathcal{W}_1} Q_w(y_w)$, so $q[z''] + \sum_{w \in \mathcal{W}_1} Q_w(y_w) = P(\widehat{A}'') - P(B'')$. Now the goal is to show that $(\widehat{A}'', B'')$ indeed survives in the solution of the HintedKnapsackExtend$^+$ instance, i.e., for $\hat{\imath} := W(\widehat{A}'') - W(B'')$ we want to show $r[\hat{\imath}] = q[z''] + \sum_{w \in \mathcal{W}_1} Q_w(y_w)$.

First we verify that $\hat{\imath}$ is contained in the index range $[-L_j\,..\,L_j]$ of the returned table: by Lemma 3.7 (Item 2) we have $W(\widehat{A}'') \le |A' \cap J^+_{\le j}| \cdot w_{\max} \le m_j w_{\max} = L_j$, and similarly $W(B'') \le L_{j-1}$, so $|\hat{\imath}| = |W(\widehat{A}'') - W(B'')| \le L_j$ as desired. Now, if $r[\hat{\imath}] \ne q[z''] + \sum_{w \in \mathcal{W}_1} Q_w(y_w)$, then by the definition of Problem 1 there are only two possibilities:

- Case 1: $(\boldsymbol{y}, z'')$ is not a maximizer for $r[\hat{\imath}]$.

  This means there is a solution $(\boldsymbol{x}^\star, z^\star)$ with strictly better total profit $q[z^\star] + \sum_{w \in \mathcal{W}_1} Q_w(x^\star_w) > q[z''] + \sum_{w \in \mathcal{W}_1} Q_w(y_w)$. Let $X^\star \subseteq J^+_j$ be the item set encoded by $\boldsymbol{x}^\star$, and let $q[z^\star]$ correspond to the partial solution $(A^\star, B^\star)$ supported on $(J^+_{\le j-1} \cup J^-_{\le j-1})$. Then the partial solution $(A^\star \cup X^\star, B^\star)$ has the same weight as $(\widehat{A}'', B'')$ but achieves strictly higher profit (both of them are supported on $J^+_{\le j} \cup J^-_{\le j-1}$). This contradicts the $I_{\mathcal{W}_1}$-optimality of (A', B') by an exchange argument.

- Case 2: $(\boldsymbol{y}, z'')$ is a maximizer for $r[\hat{\imath}]$, but there is also another maximizer $(\boldsymbol{x}^\star, z^\star)$ for $r[\hat{\imath}]$ that violates the support containment condition $\mathrm{supp}(\boldsymbol{x}^\star) \subseteq S[z^\star] = S^+[z^\star]$ (Eq. (17)).

  Let $X^\star \subseteq J_j^+$ and $(A^\star, B^\star)$ be defined in the same way as in Case 1. Similarly to the discussion in Case 1, here we know that there is an alternative $I_{\mathcal{W}_1}$-optimal partial solution $(\widetilde{A}', \widetilde{B}') := \big((A' \setminus \widehat{A}'') \cup (A^\star \cup X^*),\ (B' \setminus B'') \cup B^\star\big)$ that achieves the same weight and profit as (A', B').

  By the violation of Eq. (17) we have $\mathrm{supp}(\mathrm{weights}(X^\star)) = \mathrm{supp}(\boldsymbol{x}^\star) \not\subseteq S^+[z^\star]$, that is, $X^\star \not\subseteq I_{S^+[z^\star]}$. Since $\widetilde{A}' \setminus (J^+_{\le j-1} \cup J^-_{\le j-1}) \supseteq X^\star$, this means $\widetilde{A}' \setminus (J^+_{\le j-1} \cup J^-_{\le j-1}) \not\subseteq I_{S^+[z^\star]}$. This violates Item 2 of Definition 3.8 for entry $q[z^\star]$ and the $I_{\mathcal{W}_1}$-optimal partial solution $(\widetilde{A}', \widetilde{B}')$, and hence contradicts the assumption that $q[\,]$ (with hints $S^+[\,], S^-[\,]$) is a hinted-$(J^+_{\le j-1} \cup J^-_{\le j-1})$-optimal DP table.

This finishes the proof that $r[\hat{\imath}] = q[z''] + \sum_{w \in \mathcal{W}_1} Q_w(y_w)$, meaning that $r[-L_j \mathinner{..} L_j]$ is indeed an $(J^+_{\le j} \cup J^-_{\le j-1})$-optimal DP table.

For the rest of the proof, without loss of generality we assume $(\boldsymbol{x}[\hat{\imath}], z[\hat{\imath}]) = (\boldsymbol{y}, z'')$ (since we could have started the proof with the $I_{\mathcal{W}_1}$-optimal partial solution (A', B') which would produce a $\boldsymbol{y}$ that coincides with $\boldsymbol{x}[\hat{\imath}]$).

New hints for hinted-optimality. Now we describe how to compute new hint arrays $T^+[-L_j \mathinner{..} L_j]$, $T^-[-L_j \mathinner{..} L_j]$ to annotate to the DP table $r[-L_j \mathinner{..} L_j]$.

We will crucially use Item 3 of Lemma 3.7, which states that for every $I_{\mathcal{W}_1}$-optimal partial solution (A', B'),

$$|\{w \in \mathcal{W}_1 : I_{\{w\}} \cap J_j^+ \subseteq A' \text{ and } I_{\{w\}} \cap J_{j+1}^+ \neq \varnothing\}| \le b_{j+1}. \tag{18}$$

Given the solutions $(\boldsymbol{x}[i], z[i])$ returned by Theorem 3.11 (where each $\boldsymbol{x}[i]$ encodes an item subset of J_j^+), we do the following for every $-L_j \le i \le L_j$:

- Define hints

$$\begin{aligned} T^-[i] &:= S^-\big[z[i]\big], \\ T^+[i] &:= \{w \in \mathcal{W}_1 : |I_{\{w\}} \cap J_j^+| = x[i]_w \text{ and } I_{\{w\}} \cap J_{j+1}^+ \neq \varnothing\}. \end{aligned}$$

- If $|T^+[i]| > b_{j+1}$, then we remove entry $r[i]$ from the DP table by setting $r[i] = -\infty$, and leave $T^-[i], T^+[i]$ undefined.

This clearly satisfies the requirement of hint size: the positive hint size is $|T^+[i]| \le b_{j+1}$, and the negative hint size is $|T^-[i]| = \big|S^-\big[z[i]\big]\big| \le b_j$. From Eq. (18), we also know that we did not remove the entry $r[\hat{\imath}]$ corresponding to our $(J^+_{\le j} \cup J^-_{\le j-1})$-optimal solution $(\widehat{A}'', B'')$ defined earlier (this is because $|I_{\{w\}} \cap J_j^+| = x[\hat{\imath}]_w = y_w$ would imply $I_{\{w\}} \cap J_j^+ \subseteq Y \subseteq A'$).

Finally we show that $T^+[\hat{\imath}]$ and $T^-[\hat{\imath}]$ satisfy Item 2 of Definition 3.8 for the DP table entry $r[\hat{\imath}]$. Suppose for contradiction that there is an $I_{\mathcal{W}_1}$-optimal partial solution (A^*, B^*) whose restriction $(\widehat{A}^{**}, B^{**})$ where $\widehat{A}^{**} := A^* \cap J^+_{\le j}, B^{**} := B^* \cap J^-_{\le j-1}$ has weight $W(\widehat{A}^{**}) - W(B^{**}) = \hat{\imath}$, such that either $A^* \setminus J^+_{\le j} \not\subseteq I_{T^+[\hat{\imath}]}$ or $B^* \setminus J^-_{\le j-1} \not\subseteq I_{T^-[\hat{\imath}]}$. Recall that $(A'' \cup Y, B'')$ (where $A'' \subseteq J^+_{\le j-1}, Y \subseteq J_j^+, B'' \subseteq J^-_{\le j-1}$) has the same weight $\hat{\imath}$ and profit $r[\hat{\imath}]$ as $(\widehat{A}^{**}, B^{**})$ does, so

$$(\widetilde{A}^*, \widetilde{B}^*) := \big((A'' \cup Y) \cup (A^* \setminus \widehat{A}^{**}),\ B'' \cup (B^* \setminus B^{**})\big) \tag{19}$$

is also an $I_{\mathcal{W}_1}$-optimal partial solution with the same weight and profit as (A^*, B^*). Now consider two cases:

- Case $A^* \setminus J^+_{\le j} \not\subseteq I_{T^+[\hat{\imath}]}$:

  Let $a \in A^* \setminus J^+_{\le j}$ and $a \notin I_{T^+[\hat{\imath}]}$. By $a \notin I_{T^+[\hat{\imath}]}$ and the definition of $T^+[\hat{\imath}]$, we know either $|I_{\{w_a\}} \cap J^+_j| > x[\hat{\imath}]_{w_a}$ or $I_{\{w_a\}} \cap J^+_{j+1} = \varnothing$. The latter is impossible given the existence of $a \in A^* \setminus J^+_{\le j} \subseteq (J^+_{j+1} \cup \cdots \cup J^+_k)$ (by Definition 3.5), so we must have $|I_{\{w_a\}} \cap J^+_j| > x[\hat{\imath}]_{w_a}$, which means some item $a' \in I_{\{w_a\}} \cap J^+_j$ is not included in the set $Y \subseteq J^+_j$ that encodes $\boldsymbol{x}[\hat{\imath}]$. Hence, the $I_{\mathcal{W}_1}$-optimal partial solution $(\widetilde{A}^*, \widetilde{B}^*)$ in Eq. (19) contains a but not a'. Since $\mathrm{rank}(a') < \mathrm{rank}(a)$ and $w_{a'} = w_a$, this violates the prefix property (Lemma 2.4) and hence contradicts the $I_{\mathcal{W}_1}$-optimality of $(\widetilde{A}^*, \widetilde{B}^*)$.

- Case $B^* \setminus J^-_{\le j-1} \not\subseteq I_{T^-[\hat{\imath}]}$:

  By definition, $T^-[\hat{\imath}] = S^-\left[z[\hat{\imath}]\right] = S^-[z'']$ where $z'' = W(A'') - W(B'')$. Note that the $I_{\mathcal{W}_1}$-optimal partial solution $(\widetilde{A}^*, \widetilde{B}^*)$ in Eq. (19) satisfies $W(\widetilde{A}^* \cap J^+_{\le j-1}) - W(\widetilde{B}^* \cap J^-_{\le j-1}) = W(A'') - W(B'') = z''$, so by the assumption that $q[\,]$ (with hints $S^-[\,], S^+[\,]$) is hinted-$(J^+_{\le j-1} \cup J^-_{\le j-1})$-optimal, we must have $\widetilde{B}^* \setminus J^-_{\le j-1} \subseteq I_{S^-[z'']}$ by Definition 3.8 (Item 2). However, $\widetilde{B}^* \setminus J^-_{\le j-1} = B^* \setminus J^-_{\le j-1} \not\subseteq I_{T^-[\hat{\imath}]} = I_{S^-[z'']}$, a contradiction.

Hence, we have verified that $T^+[\hat{\imath}]$ and $T^-[\hat{\imath}]$ satisfy Item 2 of Definition 3.8 for the DP table entry $r[\hat{\imath}]$, so $r[\,]$ (with hints $T^+[\,], T^-[\,]$) is indeed a hinted-$(J^+_{\le j} \cup J^-_{\le j-1})$-optimal DP table. This finishes the proof of the correctness of our algorithm.

The time complexity of applying Theorem 3.11 is $O(L_j b_j \log^2(L_j b_j))$, and the time complexity of preprocessing functions $Q_w(\cdot)$ is $O(|J^+_j| + |\mathcal{W}_1| w_{\max} \log w_{\max})$. Since $|\mathcal{W}_1| \le O((w_{\max} \log w_{\max})^{1/2})$ and $L_j b_j = \Theta(w^2_{\max} \log w_{\max})$ (by Eqs. (12) and (13)), the total running time is $O(L_j b_j \cdot \log^2(L_j b_j) + |J^+_j|)$. □

4 Algorithm for HintedKnapsackExtend$^+$

In this section we describe our algorithm for the HintedKnapsackExtend$^+$ problem (Problem 1), proving Theorem 3.11. In Lemma 4.1, we solve the special case where the hints are singleton sets. Then in Section 4.2, we provide several helper lemmas that allow us to decompose an instance into multiple instances with smaller hint sets. Finally in Section 4.3 we put the pieces together to solve the general case.

4.1 The base case with singleton hint sets

The following lemma is the most interesting building block of our algorithm for Problem 1.

Lemma 4.1. *HintedKnapsackExtend$^+$ (Problem 1) with $|S[i]| \le 1$ for all $-L \le i \le L$ can be solved deterministically in $O(L \log L)$ time.*

More precisely, the algorithm runs in $O(L + L_1 \log L)$ time, where $L_1 = \{-L \le i \le L : S[i] \neq \varnothing\}$.

The pseudocode of our algorithm for Lemma 4.1 is given in Algorithm 1. Here we first provide an overview. Algorithm 1 contains two stages:

Algorithm 1: Solving HintedKnapsackExtend$^+$ with singleton hint sets

Input: $q[-L\,..\,L]$ and $S[-L\,..\,L]$, where $S[i] \subseteq [w_{\max}], |S[i]| \le 1, q[i] \in \mathbb{Z} \cup \{-\infty\}$ for all i

Output: $(\boldsymbol{x}[-L\,..\,L], z[-L\,..\,L], r[-L\,..\,L])$ as a solution to Problem 1

SMAWKAndScan$(q[-L\,..\,L], S[-L\,..\,L])$:

begin

`/* Stage 1: use SMAWK to find all candidate updates` $q[j] + Q_w(\frac{i-j}{w})$ `where` $w \in S[j]$`, expressed as difference-`w `APs consisting of indices` i `*/`

Initialize $\mathcal{P} \leftarrow \varnothing$

for $w \in [w_{\max}]$ *and* $c \in \{0, 1, \ldots, w-1\}$ **do**

$J := \{j : w \in S[j] \text{ and } j \equiv c \pmod w, -L \le j \le L\}$

$I := \{i : i \equiv c \pmod w, -L \le i \le L\}$

Run SMAWK (Theorem A.1) on matrix $A_{I \times J}$ defined as $A[i,j] := q[j] + Q_w\left(\frac{i-j}{w}\right)$.

for $j \in J$ **do**

Suppose SMAWK returned the AP $P_j \subseteq I$ of difference w, such that for every $i \in P_j$, $j = \arg\max_{j' \in J} A[i, j']$

$P_j \leftarrow P_j \cap \{i \in \mathbb{Z} : i > j\}$ `// focus on candidate updates where` $\frac{i-j}{w}$ `is a positive integer`

Suppose $P_j = \{c + kw, c + (k+1)w, \ldots, c + \ell w\}$, and insert $(j; c, w, k, \ell)$ into $\mathcal{P}$

`/* Stage 2: combine all candidate updates by a linear scan from left to right, extending winning APs and discarding losing APs */`

Initialize empty buckets $B[-L], B[-L+1], \ldots, B[L]$

for $(j; c, w, k, \ell) \in \mathcal{P}$ **do**

Insert $(j; c, w, k, \ell)$ into bucket $B[c + kw]$ `// insert to the bucket indexed by the beginning element of the AP`

for $i \leftarrow -L, \ldots, L$ **do**

$r[i] \leftarrow q[i], z[i] \leftarrow i, \boldsymbol{x}[i] \leftarrow \boldsymbol{0}$. `// the trivial solution for` i

if $B[i] \neq \varnothing$ **then**

Pick $(j; c, w, k, \ell) \in B[i]$ that maximizes $q[j] + Q_w\left(\frac{i-j}{w}\right)$

if $q[j] + Q_w\left(\frac{i-j}{w}\right) > r[i]$ **then**

$r[i] \leftarrow q[j] + Q_w\left(\frac{i-j}{w}\right), z[i] \leftarrow j, \boldsymbol{x}[i] \leftarrow \frac{i-j}{w}\boldsymbol{e}_w$. `// solution for` i

if $i + w \le c + \ell w$ **then**

Insert $(j; c, w, k, \ell)$ into bucket $B[i + w]$ `// extend this winning AP by one step, and all other APs in the bucket` $B[i]$ `are discarded`

return $(\boldsymbol{x}[-L\,..\,L], z[-L\,..\,L], r[-L\,..\,L])$

- In the first stage, we enumerate $w \in [w_{\max}]$ and $c \pmod w$, and collect indices $j \equiv c \pmod w$ such that $w \in S[j]$. Then we try to extend from these collected indices j by adding integer multiples of w (which does not interfere with other congruence classes modulo w): using SMAWK algorithm [AKM+87], for every $i \equiv c \pmod w$, find j among the collected indices to maximize $q[j] + Q_w(\frac{i-j}{w})$. This is the same idea as in the proof of the standard batch-update Lemma 2.7 (used in e.g., [KP04, Cha18, AT19, PRW21]). However, in our scenario with small sets $S[j]$, the number of collected indices j is usually sublinear in the array size L, so in order to save time we need to let SMAWK return a compact output representation, described as several arithmetic progressions (APs) with difference w, where each AP contains the indices i that have the same maximizer j.

- The second stage is to combine all the APs found in the first stage, and update them onto a single DP array. Ideally, we would like to take the entry-wise maximum over all the APs, that is, for each i we would like to maximize $q[j] + Q_w(\frac{i-j}{w})$ over all APs containing i, where w is the difference of the AP and j is the maximizer associated to that AP. Unfortunately, the total length of these APs could be much larger than the array size L, which would prevent us from getting an $\widetilde{O}(L)$ time algorithm. To overcome this challenge, the idea here is to crucially use the relaxation in the definition of Problem 1, so that we can skip a lot of computation based on the concavity of $Q_w(\cdot)$. We perform a linear scan from left to right, and along the way we discard many APs that cannot contribute to any useful answers. In this way we can get the time complexity down to near-linear.

Proof of Lemma 4.1. The algorithm is given in Algorithm 1.

Time complexity. We first analyze the time complexities of the two stages of Algorithm 1.

- The first stage contains a **for** loop over $w \in [w_{\max}]$ and $c \in \{0, 1 \dots, w-1\}$ (Line 4), but we actually only need to execute the loop iterations such that the index set $J := \{j : w \in S[j] \text{ and } j \equiv c \pmod w, -L \le j \le L\}$ (defined at Line 5) is non-empty. Since $|S[j]| \le 1$ for all j, these sets J over all (w, c) form a partition of the size-L_1 set $\{-L \le j \le L : S[j] \neq \varnothing\}$, and can be prepared efficiently. Then, for each of these sets J, at Line 7 we run SMAWK algorithm (Theorem A.1) to find all row maxima (with compact output representation) of an $O(1 + L/w) \times |J|$ matrix in $O(|J| \log L)$ time. The output of SMAWK is represented as $|J|$ intervals on the row indices of this matrix, which correspond to $|J|$ APs of difference w. These $|J|$ APs are then added into $\mathcal{P}$. Thus, in the end of the first stage, set $\mathcal{P}$ contains at most $\sum_J |J| \le L_1 \le 2L + 1$ APs (each AP only takes $O(1)$ words to describe), and the total running time of this stage is $O(\sum_J |J| \log L) = O(L_1 \log L)$.

- In the second stage, we initialize $(2L + 1)$ buckets $B[-L \,..\, L]$, and first insert each AP from $\mathcal{P}$ into a bucket (Line 13). Then we do a scan $i \leftarrow -L, \dots, L$ (Line 15), where for each i we examine all APs in the bucket $B[i]$ at Line 18, and then copy at most one winning AP from this bucket into another bucket (Line 22). Hence, in total we only ever inserted at most $|\mathcal{P}| + (2L + 1) = O(L)$ APs into the buckets. So the second stage takes $O(L)$ overall time.

Hence the total time complexity of Algorithm 1 is $O(L_1 \log L + L) \le O(L \log L)$.

Correctness. It remains to prove that the return values $(\boldsymbol{x}[i], z[i], r[i])$ correctly solve Problem 1. Fix any $i \in \{-L, \dots, L\}$, and let $(\boldsymbol{x}^*[i], z^*[i], r^*[i])$ be an maximizer of Eq. (15) (subject to Eq. (16)). If $\boldsymbol{x}^*[i] = \mathbf{0}$, then it is the trivial solution, which cannot be better than our solution, due to Lines 16 and 19. So in the following we assume $|\mathrm{supp}(\boldsymbol{x}^*[i])| \geq 1$, which means $i > z^*[i]$. If the support containment condition $\mathrm{supp}(\boldsymbol{x}^*[i]) \subseteq S\big[z^*[i]\big]$ (Eq. (17)) is violated, then by definition of Problem 1 we are not required to find a maximizer for i. Hence, we can assume $\mathrm{supp}(\boldsymbol{x}^*[i]) \subseteq S\big[z^*[i]\big]$ holds. Since $\big|S\big[z^*[i]\big]\big| \leq 1 \leq |\mathrm{supp}(\boldsymbol{x}^*[i])|$, we assume $\mathrm{supp}(\boldsymbol{x}^*[i]) = S\big[z^*[i]\big] = \{w^*\}$.

In the **for** loop iteration of the first stage where $w = w^*$ and $c = i \bmod w$, we have $z^*[i] \in J$ and $i \in I$. The input matrix $A_{I\times J}$ to SMAWK encodes the objective values of extending from j by adding multiples of w^*; in particular, $A[i, z^*[i]]$ equals our optimal objective $r^*[i] = q\big[z^*[i]\big] + Q_{w^*}\left(\frac{i-z^*[i]}{w^*}\right)$. So SMAWK correctly returns an AP $P_{z^*[i]} = \{c + kw^*, c + (k+1)w^*, \dots, c + \ell w^*\}$ that contains i (unless there is a tie $A[i, z^*[i]] = A[i, j]$ for some other $j \in J$, and i ends up in the AP P_j, but in this case we could have started the proof with $(\boldsymbol{x}^*[i], z^*[i])$ being this alternative maximizer $z^*[i] \leftarrow j$ and $\boldsymbol{x}^*[i] \leftarrow \frac{i-z^*[i]}{w^*}\boldsymbol{e}_{w^*}$). Since $i > z^*[i]$, we know i is not removed from $P_{z^*[i]}$ at Line 10. This AP $P_{z^*[i]}$ containing i is then added to $\mathcal{P}$.

In the second stage, each AP in $\mathcal{P}$ starts in the bucket indexed by the leftmost element of this AP (Line 13), and during the left-to-right linear scan this AP may win over others in its current bucket (at Line 18) and gets advanced to the bucket corresponding to its next element in the AP (at Line 22), or it may lose at Line 18 and be discarded. (Note that any AP can only appear in buckets whose indices belong to this AP.) Our goal is to show that the AP $P_{z^*[i]}$ can survive the competitions and arrive in bucket $B[i]$, so that it can successfully update the answer for i at Line 20.

Suppose for contradiction that $P_{z^*[i]}$ lost to some other AP $P'_{j'}$ at Line 18 when they were both in bucket $B[i_0]$ (for some $i_0 < i$). Suppose this AP $P'_{j'}$ has common difference w', and corresponds to the objective value $q[j'] + Q_{w'}\left(\frac{i'-j'}{w'}\right)$ for $i' \in P'_{j'}$. Note that $i_0 \in P'_{j'}$ satisfies $i_0 > j'$ due to Line 10. Note that $w' \neq w^*$ must hold, since two APs produced in stage 1 with the same common difference cannot intersect (because SMAWK (Theorem A.1) returns disjoint intervals), and hence cannot appear in the same bucket $B[i_0]$. Now we consider an alternative solution for index i defined as

$$(\boldsymbol{x}', j') := \left(\tfrac{i_0-j'}{w'}\boldsymbol{e}_{w'} + \tfrac{i-i_0}{w^*}\boldsymbol{e}_{w^*}\,,\; j'\right).$$

Note that $j' + \sum_{w\in[w_{\max}]} w \cdot x'_w = i$, and it has objective value

$$\begin{aligned}
&q[j'] + \sum_{w\in[w_{\max}]} Q_w(x'_w) \\
&= q[j'] + Q_{w'}\left(\tfrac{i_0-j'}{w'}\right) + Q_{w^*}\left(\tfrac{i-i_0}{w^*}\right) \\
&\geq q\left[z^*[i]\right] + Q_{w^*}\left(\tfrac{i_0-z^*[i]}{w^*}\right) + Q_{w^*}\left(\tfrac{i-i_0}{w^*}\right) && \text{(since } P_{j'} \text{ wins over } P_{z^*[i]} \text{ in bucket } B[i_0]\text{)} \\
&\geq q\left[z^*[i]\right] + Q_{w^*}\left(\tfrac{i_0-z^*[i]}{w^*} + \tfrac{i-i_0}{w^*}\right) + 0 && \text{(by concavity of } Q_{w^*}(\cdot)\text{)} \\
&= r^*[i].
\end{aligned}$$

Now there are two cases:

- $q[j'] + \sum_{w\in[w_{\max}]} Q_w(x'_w) > r^*[i]$.
  This contradicts the assumption that $r^*[i]$ is the optimal objective value for index i.

- $q[j'] + \sum_{w \in [w_{\max}]} Q_w(x'_w) = r^*[i]$.

  Then, $(\boldsymbol{x}', j')$ is also a maximizer for index i, but it has support size $|\operatorname{supp}(\boldsymbol{x}')| = 2$ due to $i > i_0 > j'$ and $w' \neq w^*$, and hence violates the support containment condition $\operatorname{supp}(\boldsymbol{x}') \subseteq S[j']$ (Eq. (17)). By definition of Problem 1, we are not required to find a maximizer for index i.

Hence, we have shown that the AP $P_{z^*[i]}$ can arrive in bucket $B[i]$. This finishes the proof that Algorithm 1 correctly solves HintedKnapsackExtend$^+$ for index i. □

4.2 Helper lemmas for problem decomposition

In this section, we show several helper lemmas for the HintedKnapsackExtend$^+$ problem (Problem 1). To get some intuition, one may first consider Problem 1 without the relaxation based on hints, which is a standard dynamic programming problem that obeys some kind of composition rule: namely, if we update a DP table $q[\,]$ with items from $U_1 \cup U_2$ (for some disjoint U_1 and U_2), it should have the same effect as first updating $q[\,]$ with U_1 to obtain an intermediate DP table, and then updating this intermediate table with U_2. The goal of this section is to formulate and prove analogous composition properties for the HintedKnapsackExtend$^+$ problem, which will be useful for our decomposition-based algorithms to be described later in Section 4.3. The proofs of these properties are quite mechanical and are similar to the arguments in the proof of Lemma 3.3 in Section 3.3, and can be safely skipped at first read.

Using the notations from Problem 1, we denote an instance of HintedKnapsackExtend$^+$ as

$$K = \big(U, \{Q_w\}_{w \in U}, S[-L\,..\,L], q[-L\,..\,L]\big),$$

where $S[i] \subseteq U$ for all i. And we denote a solution to K as

$$Y = (\boldsymbol{x}[-L\,..\,L], z[-L\,..\,L], r[-L\,..\,L]).$$

where $z[i] = i - \sum_{w \in U} w \cdot x[i]_w$ by Eq. (16), and

$$r[i] := q\big[z[i]\big] + \sum_{w \in V} Q_w(x[i]_w)$$

is the objective value. We say a solution *correctly solves* K if it satisfies the definition of Problem 1 (with the relaxation based on hints $S[\,]$).

Now we define several operations involving the instance K. In the following we omit the array index range $[-L\,..\,L]$ for brevity.

Definition 4.2 (Restriction)**.** The *restriction* of instance K to a set $V \subseteq U$ is defined as the instance

$$K|_V := \big(V, \{Q_w\}_{w \in V}, S_V[\,], q[\,]\big)$$

where $S_V[i] := S[i] \cap V$.

Definition 4.3 (Updating)**.** Suppose $Y_V = (\boldsymbol{x}[\,], z[\,], r[\,])$ is a solution to $K|_V$, then we define the following updated instance

$$K^{(V \leftarrow Y_V)} := (U \setminus V, \{Q_w\}_{w \in U \setminus V}, S'[\,], q'[\,])$$

where

$$S'[i] := S\big[z[i]\big] \setminus V, \tag{20}$$

and

$$q'[i] := r[i].$$

Definition 4.4 (Composition)**.** Let $V, V' \subseteq U, V \cap V' = \varnothing$. Suppose $Y_V = (\boldsymbol{x}[\,], z[\,], r[\,])$ is a solution to $K|_V$, and $Y_{V'} = (\boldsymbol{x}'[\,], z'[\,], r'[\,])$ is a solution to $K^{(V \leftarrow Y_V)}|_{V'}$. We define the following composition of solutions,

$$Y_{V'} \circ Y_V := (\boldsymbol{x}''[\,], z''[\,], r'[\,])$$

as a solution to $K|_{V \cup V'}$, where

$$z''[i] := z\big[z'[i]\big],$$
$$\boldsymbol{x}''[i] := \boldsymbol{x}'[i] + \boldsymbol{x}\big[z'[i]\big].$$

Note that $\circ$ is associative.

Now we are ready to state the composition lemma for HintedKnapsackExtend$^+$.

Lemma 4.5 (Composition lemma for HintedKnapsackExtend$^+$)**.** *Let $V, V' \subseteq U, V \cap V' = \varnothing$. If Y_V correctly solves $K|_V$, and $Y_{V'}$ correctly solves $K^{(V \leftarrow Y_V)}|_{V'}$, then $Y_{V'} \circ Y_V$ correctly solves $K|_{V \cup V'}$.*

Proof. Recall $K = \big(U, \{Q_w\}_{w \in U}, S[\,], q[\,]\big)$. Denote the solutions $Y_V = (\boldsymbol{x}[\,], z[\,], r[\,])$, $Y_{V'} = (\boldsymbol{x}'[\,], z'[\,], r'[\,])$, and $Y_{V'} \circ Y_V := (\boldsymbol{x}''[\,], z''[\,], r''[\,])$.

Suppose for contradiction that for some i, $\boldsymbol{x}''[i]$ is incorrect for the instance $K|_{V \cup V'}$. By definition of HintedKnapsackExtend$^+$, this means that all maximizers $(\bar{\boldsymbol{x}}, \bar{z})$ (where $\bar{\boldsymbol{x}} \in \mathbb{Z}_{\geq 0}^{V \cup V'}$) of the objective value

$$\bar{r} = q[\bar{z}] + \sum_{w \in V \cup V'} Q_w(\bar{x}_w)$$

(subject to $\bar{z} + \sum_{w \in V \cup V'} w \cdot \bar{x}_w = i$) should satisfy the support containment condition,

$$\operatorname{supp}(\bar{\boldsymbol{x}}) \subseteq S[\bar{z}].$$

Take any such maximizer $(\bar{\boldsymbol{x}}, \bar{z})$, and write $\bar{\boldsymbol{x}} = \bar{\boldsymbol{x}}_V + \bar{\boldsymbol{x}}_{V'}$ such that $\operatorname{supp}(\bar{\boldsymbol{x}}_V) \subseteq V$ and $\operatorname{supp}(\bar{\boldsymbol{x}}_{V'}) \subseteq V'$. Let

$$i_V := \bar{z} + \sum_{w \in V} w \cdot \bar{x}_w$$

and

$$r_V := q[\bar{z}] + \sum_{w \in V} Q_w(\bar{x}_w).$$

Note that in instance $K|_V$, $(\bar{\boldsymbol{x}}_V, \bar{z}, r_V)$ should be a valid solution for i_V with objective value r_V. We now compare it with $r[i_V]$ from the correct solution Y_V to the instance $K|_V$, and we claim that $r[i_V] = r_V$ must hold. Otherwise, by the definition of the HintedKnapsackExtend$^+$ instance $K|_V$, there can only be two possibilities:

- Case 1: $(\bar{\boldsymbol{x}}_V, \bar{z}, r_V)$ is not a maximizer solution for index i_V in $K|_V$.

  This means there is some solution $(\boldsymbol{x}^\star, z^\star, r^\star)$ for index i_V in $K|_V$ that achieves a higher objective $r^\star > r_V$. We will use an exchange argument to derive contradiction: Consider the solution $(\boldsymbol{x}^\star + \bar{\boldsymbol{x}}_{V'}, z^\star)$ to the instance $K|_{V\cup V'}$. It has the same total weight $z^\star + \sum_{w\in V} w\cdot x^\star_w + \sum_{w\in V'} w\cdot \bar{x}_w = i_V + \sum_{w\in V'} w\cdot\bar{x}_w = i$, but has a higher objective value $r^\star + \sum_{w\in V'} Q_w(\bar{x}_w) > r_V + \sum_{w\in V'} Q_w(\bar{x}_w) = \bar{r}$, contradicting the assumption that $(\bar{\boldsymbol{x}}, \bar{z})$ is a maximizer for i in instance $K|_{V\cup V'}$.

- Case 2: $(\bar{\boldsymbol{x}}_V, \bar{z}, r_V)$ is a maximizer solution for index i_V in $K|_V$, but there is another maximizer solution $(\boldsymbol{x}^\star, z^\star, r_V)$ for index i_V in $K|_V$ that does not satisfy the support containment condition $\mathrm{supp}(\boldsymbol{x}^\star) \subseteq S[z^\star]$ for instance $K|_V$.

  In this case, again consider the solution $(\boldsymbol{x}^\star + \bar{\boldsymbol{x}}_{V'}, z^\star)$ to the instance $K|_{V\cup V'}$. This time it has the same objective value $\bar{r}$, so it is a maximizer for index i in the instance $K|_{V\cup V'}$. However, since $\mathrm{supp}(\boldsymbol{x}^\star) \not\subseteq S[z^\star]$, we know $\mathrm{supp}(\boldsymbol{x}^\star + \bar{\boldsymbol{x}}_{V'}) \not\subseteq S[z^\star]$, and hence it violates the support containment condition in $K|_{V\cup V'}$, contradicting our assumption that all maximizers to index i in instance $K|_{V\cup V'}$ satisfy the support containment condition.

Hence we have established that $r[i_V] = r_V$ must hold.

Now we look at the second instance, $K^{(V\leftarrow Y_V)}|_{V'} = \big(V', \{Q_w\}_{w\in V'}, S'[\,], q'[\,]\big)$. Note that we have $q'[i_V] = r[i_V] = r_V$ by definition. Hence, $(\bar{\boldsymbol{x}}_{V'}, i_V, \bar{r})$ should be a valid solution for index i in instance $K^{(V\leftarrow Y_V)}|_{V'}$, with objective value

$$q'[i_V] + \sum_{w\in V'} Q_w(\bar{x}_w) = r_V + \sum_{w\in V'} Q_w(\bar{x}_w) = \bar{r}.$$

Now we claim that $r'[i] = \bar{r}$ (recall that $r'[\cdot]$ denotes objective values achieved by the solution $Y_{V'}$) must hold. Otherwise, by the definition of the HintedKnapsackExtend$^+$ instance $K^{(V\leftarrow Y_V)}|_{V'}$, there can only be two possibilities:

- Case 1: $(\bar{\boldsymbol{x}}_{V'}, i_V, \bar{r})$ is not a maximizer solution for index i in $K^{(V\leftarrow Y_V)}|_{V'}$.

  This means there is some solution $(\boldsymbol{x}^*, z^*, r^*)$ for index i in $K^{(V\leftarrow Y_V)}|_{V'}$ that achieves a higher objective $r^* > \bar{r}$. Then, in instance $K|_{V\cup V'}$, the solution $(\boldsymbol{x}^* + \boldsymbol{x}[z^*], z[z^*])$ has total weight $z[z^*] + \sum_{w\in V} w\cdot x[z^*]_w + \sum_{w\in V'} w\cdot x^*_w = z^* + \sum_{w\in V'} w\cdot x^*_w = i$ and total objective value $q[z[z^*]] + \sum_{w\in V} Q_w(x[z^*]_w) + \sum_{w\in V'} Q_w(x^*_w) = r[z^*] + \sum_{w\in V'} Q_w(x^*_w) = q'[z^*] + \sum_{w\in V'} Q_w(x^*_w) = r^* > \bar{r}$, contradicting the assumption that $(\bar{\boldsymbol{x}}, \bar{z})$ is a maximizer for i in instance $K|_{V\cup V'}$.

- Case 2: $(\bar{\boldsymbol{x}}_{V'}, i_V, \bar{r})$ is a maximizer solution for index i in $K^{(V\leftarrow Y_V)}|_{V'}$, but there is another maximizer solution $(\boldsymbol{x}^*, z^*, \bar{r})$ for index i in $K^{(V\leftarrow Y_V)}|_{V'}$ that does not satisfy the support containment condition $\mathrm{supp}(\boldsymbol{x}^*) \subseteq S'[z^*]$.

  Similar to Case 1, we again consider the solution $(\boldsymbol{x}^* + \boldsymbol{x}[z^*], z[z^*])$ in instance $K|_{V\cup V'}$, which has total weight i and total objective value $\bar{r}$. So it is an alternative maximizer to index i in instance $K|_{V\cup V'}$. Since $S'[z^*] = S[z[z^*]] \setminus V$ by definition (Eq. (20)), we have $\mathrm{supp}(\boldsymbol{x}^*) \not\subseteq S'[z^*] = S[z[z^*]] \setminus V$, which means $\mathrm{supp}(\boldsymbol{x}^*) \not\subseteq S[z[z^*]]$, and thus $\mathrm{supp}(\boldsymbol{x}^* + \boldsymbol{x}[z^*]) \not\subseteq S[z[z^*]]$. This contradicts our assumption that all maximizers to index i in instance $K|_{V\cup V'}$ satisfy the support containment condition.

Hence, we must have $r'[i] = \bar{r}$. This means that we indeed have found a maximizer to index i after we compose the solutions Y_V and $Y_{V'}$. So our solution for index i is actually correct for the instance $K|_{V \cup V'}$. This finishes the proof that $Y_{V'} \circ Y_V$ correctly solves $K|_{V \cup V'}$. □

Lemma 4.5 allows us to decompose an instance by partitioning the set $U \subseteq [w_{\max}]$. In addition to this, we also need another way to decompose an instance, which in some sense allows us to partition the array indices $[-L \,..\, L]$. First, we define the entry-wise maximum of two instances.

Definition 4.6 (Entry-wise maximum)**.** Given HintedKnapsackExtend$^+$ instances $K = \big(U, \{Q_w\}_{w \in U}, S[\,], q[\,]\big)$ and $K' = \big(U, \{Q_w\}_{w \in U}, S'[\,], q'[\,]\big)$, we define the following instance,

$$\big(U, \{Q_w\}_{w \in U}, S''[\,], q''[\,]\big),$$

where

$$(S''[i], q''[i]) := \begin{cases} (S[i], q[i]) & q[i] > q'[i], \\ (S'[i], q'[i]) & q[i] < q'[i], \\ (S[i] \cap S'[i], q[i]) & q[i] = q'[i], \end{cases}$$

and denote this instance by $\max(K, K')$. We naturally extend this definition to the entry-wise maximum of possibly more than two instances.

We also define the entry-wise maximum of two solutions $Y = (\boldsymbol{x}[\,], z[\,], r[\,]), Y' = (\boldsymbol{x}'[\,], z'[\,], r'[\,])$, by $\max(Y, Y') = (\boldsymbol{x}''[\,], z''[\,], r''[\,])$, where

$$(\boldsymbol{x}''[i], z''[i]) := \begin{cases} (\boldsymbol{x}[i], z[i]) & r[i] > r'[i], \\ (\boldsymbol{x}'[i], z'[i]) & \text{otherwise}, \end{cases}$$

and objective $r''[i]$ can be uniquely determined from $(\boldsymbol{x}''[i], z''[i])$. Note that $r''[i] \geq \max(r[i], r'[i])$ obviously holds.

Naturally, we have the following lemma for HintedKnapsackExtend$^+$.

Lemma 4.7 (Entry-wise maximum lemma)**.** *If Y correctly solves K, and Y' correctly solves K', then $\max(Y, Y')$ correctly solves $\max(K, K')$.*

Proof. Denote $K = \big(U, \{Q_w\}_{w \in U}, S[\,], q[\,]\big)$, $K' = \big(U, \{Q_w\}_{w \in U}, S'[\,], q'[\,]\big)$, $Y = (\boldsymbol{x}[\,], z[\,], r[\,]), Y' = (\boldsymbol{x}'[\,], z'[\,], r'[\,])$. Let $\max(K, K') = \big(U, \{Q_w\}_{w \in U}, S''[\,], q''[\,]\big)$ and $\max(Y, Y') = (\boldsymbol{x}''[\,], z''[\,], r''[\,])$.

Suppose for contradiction that for some i, $\boldsymbol{x}''[i]$ is incorrect for the instance $\max(K, K')$. By definition of HintedKnapsackExtend$^+$, this means that all maximizers $(\bar{\boldsymbol{x}}, \bar{z})$ (where $\bar{\boldsymbol{x}} \in \mathbb{Z}^U_{\geq 0}$) of the objective value

$$\bar{r} = q''[\bar{z}] + \sum_{w \in U} Q_w(\bar{x}_w)$$

(subject to $\bar{z} + \sum_{w \in U} w \cdot \bar{x}_w = i$) should satisfy the support containment condition,

$$\operatorname{supp}(\bar{\boldsymbol{x}}) \subseteq S''[\bar{z}].$$

Take any such maximizer $(\bar{\boldsymbol{x}}, \bar{z})$. Since $q''[\bar{z}] = \max(q[\bar{z}], q'[\bar{z}])$, without loss of generality we assume $q''[\bar{z}] = q[\bar{z}]$ by symmetry. Then, in instance K, $(\bar{\boldsymbol{x}}, \bar{z})$ is a valid solution for index i, achieving objective value $q[\bar{z}] + \sum_{w \in U} Q_w(\bar{x}_w) = \bar{r}$. If in solution Y, the found objective $r[i]$ satisfies $r[i] \geq$

$\bar{r}$, then in the entry-wise maximum solution we would have $r''[i] \geq \max(r[i], r'[i]) \geq \bar{r}$, which contradicts the assumption that $\boldsymbol{x}''[i]$ is incorrect. Hence, $r[i] < \bar{r}$ holds. Then, since Y is a correct solution to instance K, we know from the definition of HintedKnapsackExtend$^+$ instance K that there can only be two possibilities:

- Case 1: $(\bar{\boldsymbol{x}}, \bar{z})$ is not a maximizer solution for index i in K.

  This would immediately mean that the actual maximum objective for index i in instance $\max(K, K')$ is also greater than $\bar{r}$, a contradiction.

- Case 2: $(\bar{\boldsymbol{x}}, \bar{z})$ is a maximizer solution for index i in K, but there is another maximizer solution $(\boldsymbol{x}^*, z^*)$ for index i that does not satisfy the support containment condition $\text{supp}(\boldsymbol{x}^\star) \subseteq S[z^\star]$ for instance K.

  Then, there are two cases:

  - Case (2i): $q[z^*] < q'[z^*]$.
    Then, in instance $\max(K, K')$, $(\boldsymbol{x}^*, z^*)$ actually achieves a higher objective $\bar{r} + q'[z^*] - q[z^*]$, a contradiction.
  - Case (2ii): $q[z^*] \geq q'[z^*]$.
    Then, in instance $\max(K, K')$, by definition we have $S''[z^*] \subseteq S[z^*]$. Hence, $(\boldsymbol{x}^*, z^*)$ is a maximizer solution for index i in instance $\max(K, K')$ that does not satisfy the support containment condition $\text{supp}(\boldsymbol{x}^*) \subseteq S''[z^*]$, a contradiction.

Hence, we have reached contradictions in all cases. This means $\max(Y, Y')$ is a correct solution to $\max(K, K')$. □

4.3 Decomposing the problem via color-coding

In Section 4.1, we solved HintedKnapsackExtend$^+$ (Problem 1) in the base case where each hint set $S[i]$ has size at most 1. In this section, we extend it to the case with larger size bound $|S[i]| \leq b$. This is achieved by using the color-coding technique [AYZ95] to isolate the elements in the sets $S[i]$, and using the helper lemmas from Section 4.2 to combine the solutions for different color classes. Here we apply the two-level color-coding approach previously used by Bringmann [Bri17] in his near-linear time randomized subset sum algorithm.[10]

The following Lemma 4.8 gives an algorithm for HintedKnapsackExtend$^+$ which is suitable for b slightly larger than 1 (for example, polylogarithmic).

Lemma 4.8 (Algorithm for small b)**.** *HintedKnapsackExtend$^+$ (Problem 1) with $|S[i]| \leq b$ for all $-L \leq i \leq L$ can be solved deterministically in $O(Lb^2(b + \log L))$ time.*

Lemma 4.8 will be proved later in this section via color-coding. Using Lemma 4.8 as a building block, we can solve HintedKnapsackExtend$^+$ for even larger b, using another level of color-coding. The derandomized version of this color-coding is summarized in the following lemma.

[10] An advantage in our scenario is that the sets $S[i]$ to be isolated are already given to us as input, so we can derandomize the color-coding technique, whereas in Bringmann's subset sum algorithm [Bri17] the set to be isolated is the unknown solution set. Derandomizing Bringmann's algorithm is an important open problem.

Lemma 4.9 (Deterministic balls-and-bins)**.** *Given integer r, and m sets $S_1, S_2, \ldots, S_m \subseteq [n]$ such that $|S_i| \le r \log_2(2m)$ for all i, there is an $O(mr \log m \log r + n \log r)$-time deterministic algorithm that finds an r-coloring $C\colon [n] \to [r]$, such that for every $i \in [m]$ and every color $c \in [r]$,*

$$|\{j \in S_i : C(j) = c\}| \le O(\log m).$$

Lemma 4.9 follows from the results of [Rag88, Spe87] on deterministic set balancing using the method of pessimistic estimators. A proof of Lemma 4.9 is included in Appendix B.4.

Now we can prove our main Theorem 3.11, restated below.

Theorem 3.11. *HintedKnapsackExtend$^+$ (Problem 1) with $|S[i]| \le b$ for all $-L \le i \le L$ can be solved deterministically in $O(Lb \log^2(Lb))$ time.*

Proof of Theorem 3.11 assuming Lemma 4.8. If $b \le O(\log L)$, then we can directly invoke Lemma 4.8 in $O(Lb^2(b + \log L)) \le O(Lb \log^2 L)$ time. Hence, in the following we assume $b > 2\log_2(4L+2)$. Define parameter $r = b/\log_2(4L+2) > 2$, and use Lemma 4.9 to construct in $O(Lr \log L \log r + |U| \log r)$ time a coloring $h\colon U \to [r]$, such that for every $i \in \{-L, \ldots, L\}$ and every color $c \in [r]$, $S[i] \cap h^{-1}(c) \le b_1$ for some $b_1 = O(\log L)$.

Then, we iteratively apply the algorithm from Lemma 4.8 with size bound b_1, to solve for each color class $U_c := h^{-1}(c)$ ($c \in [r]$). More precisely, let K denote the input HintedKnapsackExtend$^+$ instance, and starting with instance $K_1 := K$, we iterate $c \leftarrow 1, 2, \ldots, r$, and do the following (using notations from Section 4.2):

- Solve the restricted instance $(K_c)|_{U_c}$ using Lemma 4.8 (with size bound b_1), and obtain solution Y_c.
- Define the updated instance $K_{c+1} := (K_c)^{(U_c \leftarrow Y_c)}$.

Finally, return the composed solution $Y := Y_r \circ \cdots \circ Y_2 \circ Y_1$. By inductively applying Lemma 4.5, we know Y is a correct solution to K.

We remark on the low-level implementation of the procedure described above. When we construct the new input instance for the next iteration (namely, $(K_c)^{(U_c \leftarrow Y_c)}$), we need to prepare the new input sets $S'[\cdot] \subseteq (U_{c+1} \cup \cdots \cup U_r)$, based on the current input sets $S[\cdot] \subseteq (U_c \cup \cdots \cup U_r)$ and the current solution $z[\cdot]$, according to Eq. (20). However, each set $S[i]$ may have size as large as b, so it would be to slow to copy them explicitly. In contrast, note that an instance $(K_c)|_{U_c}$ only asks for sets $S[i] \cap U_c$ as input, which have much smaller size $b_1 = O(\log m)$. So the correct implementation should be as follows: at the very beginning, each of the sets $S[-L], \ldots, S[L]$ receives an integer handle, and when we need to copy the sets we actually only pass the handles. And, given the handle of a set $S[i]$ and a color class U_c, we can report the elements in $S[i] \cap U_c$ in $O(b_1) = O(\log m)$ time (because we can preprocess these intersections at the very beginning). In this way, the time complexity for preparing the input instances to Lemma 4.8 is no longer a bottleneck.

It remains to analyze the time complexity. There are r applications of Lemma 4.8, taking $O(Lb_1^2(b_1 + \log L)) \le O(L \log^3 L)$ time each, and $O(r \cdot L \log^3 L) \le O(Lb \log^2 L)$ time in total. Combined with the time complexity of deterministic coloring at the beginning, the total time is $O(Lr \log L \log r + |U| \log r + Lb \log^2 L) \le O(Lb \log r + |U| \log r + Lb \log^2 L)$. Here we can assume $|U| \le \sum_{-L \le i \le L} |S[i]| \le O(Lb)$ without loss of generality. Hence, the total time becomes $O(Lb \log r + Lb \log^2 L) \le O(Lb \log^2(Lb))$. ∎

It remains to describe the $O(Lb^2(b+\log L))$-time algorithm claimed in Lemma 4.8. It relies on the following birthday-paradox-type color-coding lemma, which is derandomized using a standard application of pessimistic estimators:

Lemma 4.10. *Given m sets $S_1, S_2, \ldots, S_m \in [n]$ with $|S_i| \le b$, there is a deterministic algorithm in $O(n \log m + mb^3)$ time that computes $k \le \log_2(2m)$ colorings $h_1, h_2, \ldots, h_k \colon [n] \to [b^2]$, such that for every $i \in [m]$ there exists an h_j that assigns distinct colors to elements of S_i.*

We include a proof of Lemma 4.10 in Appendix B.5. Now we prove Lemma 4.8.

Proof of Lemma 4.8. Without loss of generality, assume $|U| \le \sum_{-L \le i \le L} |S[i]| \le O(Lb)$. We apply Lemma 4.10 to the sets $S[-L], S[-L+1] \ldots, S[L] \subseteq U$, and in $O(|U| \log L + Lb^3) \le O(Lb(b^2 + \log L))$ time obtain $k = O(\log L)$ colorings $h_1, \ldots, h_k \colon U \to [b^2]$ such that every set $S[i]$ is isolated by some h_j (i.e., $S[i]$ receives distinct colors under coloring h_j).

Now, based on the input HintedKnapsackExtend$^+$ instance K, we define k new instances $K^{(1)}, \ldots, K^{(k)}$ as follows: for each $1 \le j \le k$, let

$$\mathcal{I}^{(j)} = \big\{ i \in \{-L, \ldots, L\} : \ S[i] \text{ is isolated by } h_j, \text{ but not by any } h_{j'} \ (j' < j) \big\}.$$

Then $\{\mathcal{I}^{(j)}\}_{j=1}^{k}$ form a partition of $\{-L, \ldots, L\}$. Let instance $K^{(j)} = \big(U, \{Q_w\}_{w \in U}, S^{(j)}[\,], q^{(j)}[\,]\big)$ be derived from the input instance $K = \big(U, \{Q_w\}_{w \in U}, S[\,], q[\,]\big)$ with the following modification:

$$S^{(j)}[i] := \begin{cases} S[i] & i \in \mathcal{I}^{(j)}, \\ \varnothing & \text{otherwise}, \end{cases}$$

and

$$q^{(j)}[i] := \begin{cases} q[i] & i \in \mathcal{I}^{(j)}, \\ -\infty & \text{otherwise}. \end{cases}$$

Clearly, $\max(K^{(1)}, \ldots, K^{(k)}) = K$ (see Definition 4.6), so by Lemma 4.7 it suffices to solve each $K^{(j)}$ and obtain solution $Y^{(j)}$, and finally the combined solution $Y := \max\{Y^{(1)}, \ldots, Y^{(k)}\}$ is a correct solution to K.

Now we focus on each instance $K^{(j)}$. The sets $S^{(j)}[i]$ are isolated by the coloring h_j (note that there are at most $|\mathcal{I}^{(j)}|$ many non-empty sets $S^{(j)}[i]$). So we can iteratively apply the algorithm for singletons (Lemma 4.1) to solve for each color class $U_c := h_j^{-1}(c)$ ($c \in [b^2]$), in the same fashion as in the proof of Theorem 3.11. More precisely, starting with instance $K_1 := K^{(j)}$, we iterate $c \leftarrow 1, 2, \ldots, b^2$, and do the following (using notations from Section 4.2):

- Solve the restricted instance $(K_c)|_{U_c}$ using Lemma 4.1 in $O(L + |\mathcal{I}^{(j)}| \log L)$ time, and obtain solution Y_c.
- Define the updated instance $K_{c+1} := (K_c)^{(U_c \leftarrow Y_c)}$.

Finally, return the composed solution $Y^{(j)} := Y_{b^2} \circ \cdots \circ Y_2 \circ Y_1$. By inductively applying Lemma 4.5, we know $Y^{(j)}$ is a correct solution to $K^{(j)}$.

It remains to analyze the time complexity. Each of the k instances $K^{(j)}$ is solved by b^2 applications of the singleton-case algorithm (Lemma 4.1) in $O(L + |\mathcal{I}^{(j)}| \log L)$ time each. Hence the total time complexity over all k instances is $\sum_{j=1}^{k} b^2 \cdot O(L + |\mathcal{I}^{(j)}| \log L) = O(kb^2 L + b^2 \sum_{j=1}^{k} |\mathcal{I}^{(j)}| \log L) = O(Lb^2(k + \log L)) = O(Lb^2 \log L)$. Combined with the deterministic coloring step at the beginning, the overall time complexity is $O(Lb^2 \log L + Lb(b^2 + \log L)) = O(Lb^2(b + \log L))$. ☐

Acknowledgements

I thank Ryan Williams and Virginia Vassilevska Williams for useful discussions. I thank FOCS 2023 reviewers for useful comments on the previous version of this paper.

References

[ABHS22a] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Scheduling lower bounds via AND subset sum. *J. Comput. Syst. Sci.*, 127:29–40, 2022. doi:10.1016/j.jcss.2022.01.005. 7

[ABHS22b] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. *ACM Trans. Algorithms*, 18(1):6:1–6:22, 2022. doi:10.1145/3450524. 1

[ABJ+19] Kyriakos Axiotis, Arturs Backurs, Ce Jin, Christos Tzamos, and Hongxun Wu. Fast modular subset sum using linear sketching. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 58–69. SIAM, 2019. doi:10.1137/1.9781611975482.4. 7

[AK90] Alok Aggarwal and Maria M. Klawe. Applications of generalized matrix searching to geometric algorithms. *Discret. Appl. Math.*, 27(1-2):3–23, 1990. doi:10.1016/0166-218X(90)90124-U. 37

[AKM+87] Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359. 1, 3, 10, 11, 23, 37, 38

[AT19] Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster knapsack and graph algorithms. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.19. 1, 2, 3, 5, 6, 7, 10, 18, 23, 38

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337. 29

[BC22] Karl Bringmann and Alejandro Cassis. Faster knapsack algorithms via bounded monotone min-plus-convolution. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 31:1–31:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.31. 2, 6, 7

[BC23] Karl Bringmann and Alejandro Cassis. Faster 0-1-knapsack via near-convex min-plus-convolution. *arXiv preprint arXiv:2305.01593*, 2023. To appear in ESA 2023. arXiv:2305.01593. 1

[BCD+14] David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Pătraşcu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014. doi:10.1007/s00453-012-9734-3. 7

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957. 1, 8

[BFH+22] Karl Bringmann, Nick Fischer, Danny Hermelin, Dvir Shabtay, and Philip Wellnitz. Faster minimization of tardy processing time on a single machine. *Algorithmica*, 84(5):1341–1356, 2022. doi:10.1007/s00453-022-00928-w. 7

[BFP+73] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9. 9, 10, 11

[BHSS18] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for knapsack via convolution and prediction. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1269–1282. ACM, 2018. doi:10.1145/3188745.3188876. 1

[BN21] Karl Bringmann and Vasileios Nakos. Fast n-fold boolean convolution via additive combinatorics. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 41:1–41:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.41. 6

[Bri17] Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1073–1084. SIAM, 2017. doi:10.1137/1.9781611974782.69. 1, 6, 29

[Bri23] Karl Bringmann. Knapsack with small items in near-quadratic time. *arXiv preprint arXiv:2308.03075*, 2023. arXiv:2308.03075. 2

[BW21] Karl Bringmann and Philip Wellnitz. On near-linear-time algorithms for dense subset sum. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1777–1796. SIAM, 2021. arXiv:2010.09096, doi:10.1137/1.9781611976465.107. 1, 3, 4, 6, 7, 11, 14, 40, 41, 42

[CDXZ22] Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1529–1542. ACM, 2022. doi:10.1145/3519935.3520057. 7

[CFG89] Mark Chaimovich, Gregory Freiman, and Zvi Galil. Solving dense subset-sum problems by using analytical number theory. *J. Complexity*, 5(3):271–282, 1989. doi:10.1016/0885-064X(89)90025-3. 3

[CFP21] David Conlon, Jacob Fox, and Huy Tuan Pham. Subset sums, completeness and colorings. *arXiv preprint arXiv:2104.14766*, 2021. arXiv:2104.14766. 3

[CGST86] William J. Cook, A. M. H. Gerards, Alexander Schrijver, and Éva Tardos. Sensitivity theorems in integer linear programming. *Math. Program.*, 34(3):251–264, 1986. doi:10.1007/BF01582230. 3

[CH22] Timothy M. Chan and Qizheng He. More on change-making and related problems. *J. Comput. Syst. Sci.*, 124:159–169, 2022. doi:10.1016/j.jcss.2021.09.005. 2, 4, 6, 7

[Cha99a] Mark Chaimovich. New algorithm for dense subset-sum problem. Number 258, pages xvi, 363–373. 1999. Structure theory of set addition. 3

[Cha99b] Mark Chaimovich. New structural approach to integer programming: a survey. Number 258, pages xv–xvi, 341–362. 1999. Structure theory of set addition. 3

[Cha18] Timothy M. Chan. Approximation schemes for 0-1 knapsack. In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, volume 61 of *OASIcs*, pages 5:1–5:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/OASIcs.SOSA.2018.5. 6, 23

[CL15] Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 31–40. ACM, 2015. doi:10.1145/2746539.2746568. 7

[CLMZ23] Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. Faster algorithms for bounded knapsack and bounded subset sum via fine-grained proximity results. *arXiv preprint arXiv:2307.12582v1*, 2023. arXiv:2307.12582v1. 1, 2, 3, 4, 6, 7, 9, 10, 11, 12, 14, 18, 39, 41

[CMWW19] Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. On problems equivalent to (min, +)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019. doi:10.1145/3293465. 1, 2, 6

[CW21] Timothy M. Chan and R. Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926. 7

[DJM23] Mingyang Deng, Ce Jin, and Xiao Mao. Approximating knapsack and partition via dense subset sums. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2961–2979. SIAM, 2023. arXiv:2301.09333, doi:10.1137/1.9781611977554.ch113. 3, 6

[DMZ23] Mingyang Deng, Xiao Mao, and Ziqian Zhong. On problems related to unbounded subsetsum: A unified combinatorial approach. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2980–2990. SIAM, 2023. doi:10.1137/1.9781611977554.ch114. 2, 4, 5, 6, 7, 16

[ES90] P. Erdős and A. Sárközy. On a problem of Straus. In *Disorder in physical systems*, Oxford Sci. Publ., pages 55–66. Oxford Univ. Press, New York, 1990. 41

[ES06] Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Oper. Res. Lett.*, 34(5):564–568, 2006. doi:10.1016/j.orl.2005.09.008. 4

[EW20] Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. *ACM Trans. Algorithms*, 16(1):5:1–5:14, 2020. doi:10.1145/3340322. 1, 3, 7, 11

[Fre88] Gregory A. Freiman. On extremal additive problems of Paul Erdős. *Ars Combin.*, 26(B):93–114, 1988. 3

[Fre90] G. A. Freiman. Subset-sum problem with different summands. In *Proceedings of the Twentieth Southeastern Conference on Combinatorics, Graph Theory, and Computing (Boca Raton, FL, 1989)*, volume 70, pages 207–215, 1990. 3

[Fre93] Gregory A. Freiman. New analytical results in subset-sum problem. volume 114, pages 205–217. 1993. Combinatorics and algorithms (Jerusalem, 1988). doi:10.1016/0012-365X(93)90367-3. 3

[GM91] Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. *SIAM J. Comput.*, 20(6):1157–1189, 1991. doi:10.1137/0220072. 3, 40

[HS74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21(2):277–292, 1974. doi:10.1145/321812.321823. 1

[Jin19] Ce Jin. An improved FPTAS for 0-1 knapsack. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 76:1–76:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.76. 6

[JR19] Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.43. 6

[JR22] Klaus Jansen and Lars Rohwedder. On integer programming, discrepancy, and convolution. *Mathematics of Operations Research*, 0(0):1–15, 2022. doi:10.1287/moor.2022.1308. 6

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9. 1

[Kle22] Kim-Manuel Klein. On the fine-grained complexity of the unbounded subsetsum and the frobenius problem. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3567–3582. SIAM, 2022. doi:10.1137/1.9781611977073.141. 4, 6

[KP04] Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an FPTAS for the knapsack problem. *J. Comb. Optim.*, 8(1):5–11, 2004. doi:10.1023/B:JOCO.0000021934.29833.6b. 1, 3, 5, 10, 18, 23, 38

[KPP04] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004. doi:10.1007/978-3-540-24777-7. 1

[KPR23] Kim-Manuel Klein, Adam Polak, and Lars Rohwedder. On minimizing tardy processing time, max-min skewed convolution, and triangular structured ilps. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2947–2960. SIAM, 2023. doi:10.1137/1.9781611977554.ch112. 7

[KPS17] Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.21. 1, 2, 6

[KX19] Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Trans. Algorithms*, 15(3):40:1–40:20, 2019. doi:10.1145/3329863. 1, 3

[LPV20] Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 53:1–53:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.53. 6

[MWW19] Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. A subquadratic approximation scheme for partition. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 70–88. SIAM, 2019. doi:10.1137/1.9781611975482.5. 3, 6

[Pis99] David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *J. Algorithms*, 33(1):1–14, 1999. doi:10.1006/jagm.1999.1034. 6

[PRW21] Adam Polak, Lars Rohwedder, and Karol Węgrzycki. Knapsack and subset sum with small items. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 106:1–106:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. arXiv:2105.04035, doi:10.4230/LIPIcs.ICALP.2021.106. 1, 2, 3, 5, 6, 7, 9, 10, 11, 18, 23, 38, 40

[Rag88] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *J. Comput. Syst. Sci.*, 37(2):130–143, 1988. doi:10.1016/0022-0000(88)90003-7. 30, 43

[Sár89] A. Sárközy. Finite addition theorems. I. *J. Number Theory*, 32(1):114–130, 1989. doi:10.1016/0022-314X(89)90102-9. 3, 40

[Sár94] A. Sárközy. Finite addition theorems. II. *J. Number Theory*, 48(2):197–218, 1994. doi:10.1006/jnth.1994.1062. 3, 40

[Spe87] Joel Spencer. Ten lectures on the probabilistic method. In *Proc. CBMS-NRM Regional Conference Series in Applied Mathematics*, volume 52. SIAM, 1987. 30, 43

[SS81] Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM Journal on Computing*, 10(3):456–464, 1981. doi:10.1137/0210033. 1

[SV06] E. Szemerédi and V. H. Vu. Finite and infinite arithmetic progressions in sumsets. *Ann. of Math. (2)*, 163(1):1–35, 2006. doi:10.4007/annals.2006.163.1. 3

[Wil18] R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524. 7

A SMAWK algorithm

We review the classic result of Aggarwal, Klawe, Moran, Shor, and Wilber [AKM+87] on finding row maxima in convex totally monotone matrices (in particular, convex Monge matrices).

We say an $m \times n$ real matrix A is *convex Monge* if

$$A[i,j] + A[i',j'] \geq A[i,j'] + A[i',j] \tag{21}$$

for all $i < i'$ and $j < j'$.

More generally, we consider matrices with possibly $-\infty$ entries. Following the terminology of [AK90], a matrix $A \in (\mathbb{R} \cup \{-\infty\})^{m \times n}$ is called a *reverse falling staircase* matrix, if the finite entries in each row form a prefix, and the finite entries in each column form a suffix. In other words, if $A[i,j'] > -\infty$, then $A[i',j] > -\infty$ for all $i' \in \{i, i+1, \ldots, m\}$ and $j \in \{1, 2, \ldots, j'\}$. In the definition of convex Monge property, inequality (21) is always considered to hold when its right-hand side evaluates to $-\infty$.

A reverse falling staircase matrix A is convex Monge implies A is *convex totally monotone*: for all $i < i'$ and $j < j'$,

$$A[i,j] < A[i,j'] \Rightarrow A[i',j] < A[i',j']. \tag{22}$$

Given an $m \times n$ convex totally monotone matrix, let $j_{\max}(i)$ denote the index of the leftmost column containing the maximum value in row i. Note that condition (22) implies

$$1 \leq j_{\max}(1) \leq j_{\max}(2) \leq \cdots \leq j_{\max}(m) \leq n. \tag{23}$$

The SMAWK algorithm [AKM+87] finds $j_{\max}(i)$ for all $1 \leq i \leq m$. On tall matrices ($n \ll m$), its time complexity is near-linear in n (instead of m), if we allow a compact output representation based on (23). This is formally summarized in the following theorem.

Theorem A.1 (SMAWK algorithm [AKM+87]). *Let an $m \times n$ convex Monge reverse falling staircase matrix A be implicitly given, so that each entry of A can be accessed in constant time.*

There is a deterministic algorithm that finds all row maxima of A in $O\left(n(1+\log\lceil\frac{m}{n}\rceil)\right)$ time. Its output is compactly represented as $n+1$ integers, $1 = r_1 \le r_2 \le \cdots \le r_n \le r_{n+1} = m+1$, indicating that for all $1 \le j \le n$ and $r_j \le i < r_{j+1}$, the leftmost maximum element in row i of A is $A[i,j]$.

In the context of knapsack algorithms (e.g., [KP04, AT19, PRW21]), SMAWK algorithm is used to find the $(\max,+)$-convolution $c[i] := \max_{0\le j\le i}\{a[j]+b[i-j]\}$ between an array $a[0\,..\,n]$ and a *concave* array $b[0\,..\,m]$ (that is, $b[i]-b[i-1] \ge b[i+1]-b[i]$). To do this, define matrix $A[i,j] = \begin{cases} a[j]+b[i-j] & j \le i \\ -\infty & j > i \end{cases}$. Note that A is a reverse falling staircase matrix. One can verify that A is convex Monge: for $i < i', j < j'$ such that all four terms in Eq. (21) are finite, we have

$$\begin{aligned} & A[i,j] + A[i',j'] - A[i,j'] - A[i',j] \\ = {} & b[j-i] + b[j'-i'] - b[j'-i] - b[j-i'] \\ \ge {} & 0 \end{aligned}$$

by the concavity of b. So SMAWK algorithm can compute all row maxima of A, which correspond to the answer of the $(\max,+)$-convolution.

B Omitted proofs

B.1 Proof of Lemma 2.1

Lemma 2.1 (Break ties). *Given a 0-1 Knapsack instance I, in $O(n)$ time we can deterministically reduce it to another 0-1 Knapsack instance I' with $n, w_{\max}$ and t unchanged, and $p'_{\max} \le \mathrm{poly}(p_{\max}, w_{\max}, n)$, such that the items in I' have* distinct efficiencies *and distinct profits.*

Proof. Suppose instance I has capacity t and n items $(w_1,p_1),\ldots,(w_n,p_n)$. Define instance I' with capacity t and items $(w_1,p'_1),\ldots,(w_n,p'_n)$ with modified profits

$$p'_i := (p_i \cdot M + i) \cdot w_{\max} + 1,$$

where $M := 1 + n + \sum_{i=1}^{n} i$. Then, for any item set $S \subseteq [n]$, we have

$$0 \le \sum_{i\in S} p'_i - M w_{\max} \sum_{i\in S} p_i = |S| + \sum_{i\in S} i w_{\max} < M w_{\max},$$

and hence

$$\sum_{i\in S} p_i = \left\lfloor \frac{\sum_{i\in S} p'_i}{M w_{\max}} \right\rfloor,$$

so any optimal solution for I' must also be an optimal solution for I.

For any $i \ne j$, note that $p'_i \bmod (M w_{\max}) = i w_{\max} + 1 \ne j w_{\max} + 1 = p'_j \bmod (M w_{\max})$, so $p'_i \ne p'_j$. If $p'_i/w_i = p'_j/w_j$, then from $p'_i w_j \equiv w_j \pmod{w_{\max}}$ and $p'_j w_i \equiv w_i \pmod{w_{\max}}$ we have $w_j = w_i$, which then contradicts $p'_i \ne p'_j$. So $p'_i/w_i \ne p'_j/w_j$. □

B.2 Proof of Lemma 3.2

Lemma 3.2 (Extension of [CLMZ23, Lemma 13]). *The set $\mathcal{W}$ of input item weights can be partitioned in $O(n + w_{\max} \log w_{\max})$ time into $\mathcal{W} = \mathcal{W}_1 \uplus \mathcal{W}_2 \uplus \cdots \uplus \mathcal{W}_s$, where $s < \log_2(\sqrt{w_{\max}})$, with the following property:*

Denote $\mathcal{W}_{\le j} = \mathcal{W}_1 \cup \cdots \cup \mathcal{W}_j$ and $\mathcal{W}_{>j} = \mathcal{W} \setminus \mathcal{W}_{\le j}$. For every optimal exchange solution (A, B) and every $1 \le j \le s$,

- $|\mathcal{W}_j| \le 4C\sqrt{w_{\max} \log w_{\max}} \cdot 2^j$, *and*
- $W(A \cap I_{\mathcal{W}_{>j}}) \le 4Cw_{\max}^{3/2}/2^j$ *and* $W(B \cap I_{\mathcal{W}_{>j}}) \le 4Cw_{\max}^{3/2}/2^j$,

where C is the universal constant from Lemma 3.1.

Proof. Assume the n items are sorted in decreasing order of efficiency, and the greedy solution is $G = \{1, 2, \ldots, i^*\}$. Following [CLMZ23], for each $1 \le j \le s$, we define ℓ_j to be the smallest $1 \le \ell_j \le i^*$ such that $|\operatorname{supp}(\{w_{\ell_j}, w_{\ell_j+1}, \ldots, w_{i^*}\})| \le 2C\sqrt{w_{\max} \log w_{\max}} \cdot 2^j$. Similarly, define r_j to be the largest $i^* < r_j \le n$ such that $|\operatorname{supp}(\{w_{i^*+1}, \ldots, w_{r_j}\})| \le 2C\sqrt{w_{\max} \log w_{\max}} \cdot 2^j$. Note that $[\ell_j, r_j] \subseteq [\ell_{j+1}, r_{j+1}]$. We define the number of layers s to be the smallest s such that $2C\sqrt{w_{\max} \log w_{\max}} \cdot 2^s \ge w_{\max}$. Then, in the last layer $j = s$, we have $\ell_s = 1, r_s = n$ by definition. Note that $s < \log_2(\sqrt{w_{\max}})$.

Then, the partition $\mathcal{W} = \mathcal{W}_1 \uplus \mathcal{W}_2 \uplus \cdots \uplus \mathcal{W}_s$ is defined as follows:

- Let $\mathcal{W}_{\le j} := \operatorname{supp}(\{w_{\ell_j}, w_{\ell_j+1}, \ldots, w_{r_j}\})$ for all $1 \le j \le s$.
- Then, let $\mathcal{W}_1 := \mathcal{W}_{\le 1}$, and $\mathcal{W}_j := \mathcal{W}_{\le j} \setminus \mathcal{W}_{\le j-1}$ for $2 \le j \le s$.

Since $[\ell_j, r_j] \subseteq [\ell_{j+1}, r_{j+1}]$ and $[\ell_s, r_s] = [1, n]$, this construction defines a partition $\mathcal{W} = \mathcal{W}_1 \uplus \mathcal{W}_2 \uplus \cdots \uplus \mathcal{W}_s$. By definition of ℓ_j, r_j, it automatically satisfies the first property $|\mathcal{W}_j| \le |\mathcal{W}_{\le j}| \le 4C\sqrt{w_{\max} \log w_{\max}} \cdot 2^j$ from the lemma statement. It remains to verify the second claimed property $W(A \cap I_{\mathcal{W}_{>j}}) \le 4Cw_{\max}^{3/2}/2^j$ (the other property $W(B \cap I_{\mathcal{W}_{>j}}) \le 4Cw_{\max}^{3/2}/2^j$ can be proved similarly).

Suppose this property is violated by some optimal exchange solution (A, B) (where $A \subseteq \overline{G} = \{i^* + 1, \ldots, n\}, B \subseteq G = \{1, \ldots, i^*\}$) and some $1 \le j \le s$, i.e.,

$$W(A \cap I_{\mathcal{W}_{>j}}) > 4Cw_{\max}^{3/2}/2^j. \tag{24}$$

In particular, $A \cap I_{\mathcal{W}_{>j}} \ne \varnothing$ implies $r_j \ne n$. By definition of r_j, this means

$$|\operatorname{supp}(\{w_{i^*+1}, \ldots, w_{r_j}\})| > 2C\sqrt{w_{\max} \log w_{\max}} \cdot 2^j - 1. \tag{25}$$

Now consider the following two cases. Recall $\text{weights}(I) := \biguplus_{i \in I}\{w_i\}$ denotes the multiset of item weights in $I \subseteq [n]$.

- Case $|\operatorname{supp}(\text{weights}(A))| \ge C\sqrt{w_{\max} \log w_{\max}} \cdot 2^j$:

  Recall from Eq. (5) that $|W(A) - W(B)| < w_{\max}$, so by Eq. (24) we have

  $$W(B) > W(A) - w_{\max} \ge 4Cw_{\max}^{3/2}/2^j - w_{\max} \ge 3Cw_{\max}^{3/2}/2^j,$$

  where we used $2^j \le 2^s < \sqrt{w_{\max}}$ and assumed C is large enough. Hence, $|\operatorname{supp}(\text{weights}(A))| \cdot W(B) > 3C^2 w_{\max}^2 \sqrt{\log w_{\max}}$, and Lemma 3.1 (with $N := w_{\max}$) applied to weights(A) and weights(B) implies the existence of two non-empty item subsets $A' \subseteq A, B' \subseteq B$ with the same total weight $W(A') = W(B')$. Then, $(A \setminus A', B \setminus B')$ achieves strictly higher profit than (A, B), contradicting the optimality of (A, B).

- Case $|\operatorname{supp}(\text{weights}(A))| < C\sqrt{w_{\max}\log w_{\max}}\cdot 2^j$:

  Then, we have

$$\begin{aligned}
&\left|\operatorname{supp}\left(\text{weights}\left(\{i^*+1,\dots,r_j\}\setminus A\right)\right)\right| \\
&\geq |\operatorname{supp}(\{w_{i^*+1},\dots,w_{r_j}\})| - |\operatorname{supp}(\text{weights}(A))| \\
&> (2C\sqrt{w_{\max}\log w_{\max}}\cdot 2^j - 1) - C\sqrt{w_{\max}\log w_{\max}}\cdot 2^j \qquad \text{(by Eq. (25))} \\
&> C\sqrt{w_{\max}\log w_{\max}}\cdot 2^{j-1}.
\end{aligned}$$

  Then by Eq. (24) we have

$$\left|\operatorname{supp}\left(\text{weights}\left(\{i^*+1,\dots,r_j\}\setminus A\right)\right)\right|\cdot W(A\cap I_{\mathcal{W}_{>j}}) \geq 2C^2 w_{\max}^2\sqrt{\log w_{\max}},$$

  and we can apply Lemma 3.1 to weights $\left(\{i^*+1,\dots,r_j\}\setminus A\right)$ and weights$(A\cap I_{\mathcal{W}_{>j}})$ to obtain non-empty $A'\subseteq A\cap I_{\mathcal{W}_{>j}}, B'\subseteq\{i^*+1,\dots,r_j\}\setminus A$ with the same total weight $W(A')=W(B')$. By definition of $\mathcal{W}_{\leq j}$ we know $\min_{i\in A'} i > r_j \geq \max_{i\in B'} i$. Hence, $((A\setminus A')\cup B', B)$ achieves strictly higher profit than (A,B), contradicting the optimality of (A,B).

We have reached contradiction in both cases, so Eq. (24) cannot hold. This finishes the proof of the claimed properties of the partitioning $\mathcal{W}=\mathcal{W}_1\uplus\mathcal{W}_2\uplus\cdots\uplus\mathcal{W}_s$.

Finally we briefly describe how this partitioning can be computed without actually sorting the n items in $O(n\log n)$ time. First recall that the break point element i^* of the greedy solution can be found in deterministic $O(n)$ time using linear-time median finding algorithms as in [PRW21]. Then, for all $w\in\mathcal{W}$, we can find the item in $\{i\in[n] : w_i = w,\ p_i/w_i < p_{i^*}/w_{i^*}\}$ with the highest efficiency, as well as the item in $\{i\in[n] : w_i = w,\ p_i/w_i > p_{i^*}/w_{i^*}\}$ with the lowest efficiency, in $O(n+w_{\max})$ total time. We then sort these $2|\mathcal{W}|$ items by their efficiencies in $O(w_{\max}\log w_{\max})$ time. Using this information, we can easily compute the weights of the boundary items ℓ_j, r_j defined above, and then we can obtain the partitioning. □

B.3 Proof of Lemma 3.6

Here we show how a slightly weaker version of Lemma 3.6 with one extra $\log(2r)$ factor can be derived from the additive combinatorial result of Bringmann and Wellnitz [BW21] (which built on works of Sárközy [Sár89, Sár94] and Galil and Margalit [GM91]). In the end we will briefly explain how to remove this logarithmic factor by opening the black box of [BW21].

Lemma B.1 (Weaker version of Lemma 3.6)**.** *There is a constant C such that the following holds. Suppose two multisets A, B supported on $[N]$ satisfy*

$$|\operatorname{supp}_r(A)| \geq C\sqrt{N/r}\cdot\sqrt{\log(2N)}\log(2r) \tag{26}$$

for some $r\geq 1$, and

$$\Sigma(B)\geq\Sigma(A)-N. \tag{27}$$

Then, $\mathcal{S}^*(A)\cap\mathcal{S}^*(B)\neq\varnothing$.

To prove Lemma B.1, we need the following lemma by Bringmann and Wellnitz [BW21]. For a multiset X, denote the maximum multiplicity of X by $\mu_X := \max_x \mu_X(x)$.

Lemma B.2 ([BW21, Section 4.1]). *Let X be a non-empty multiset supported on $[N]$, and*

$$C_\delta = 1699200 \cdot \log(2|X|) \log^2(2\mu_X), \tag{28}$$

$$C_2 = 4 + 2 \cdot 169920 \cdot \log(2\mu_X). \tag{29}$$

If $|X|^2 \geq C_\delta \cdot \mu_X \cdot N$, then there exists an integer

$$1 \leq d \leq 4\mu_X \Sigma(X)/|X|^2$$

such that for any integer

$$t \in [C_2 \mu_X N \Sigma(X)/|X|^2, \Sigma(X)/2]$$

that is a multiple of d, it holds that $t \in \mathcal{S}(X)$.

Remark B.3. This statement is paraphrased from [BW21, Theorem 4.3] (see also [BW21, Definition 3.1]). The original statement there stated more generally that $t \in \mathcal{S}(X)$ if and only if $t \bmod d \in \mathcal{S}(X) \bmod d$; since $0 \in \mathcal{S}(X)$, the statement here applies in particular to t that is a multiple of d. The upper bound $d \leq 4\mu_X \Sigma(X)/|X|^2$ stated here can be found in [BW21, Theorem 4.1].

Proof of Lemma B.1. The proof uses a similar strategy as [CLMZ23] (and also [ES90]). By the assumption Eq. (26), we can pick $X \subseteq A$ such that

$$\mu_X(x) = r \text{ for all } x \in X, \tag{30}$$

$$|\operatorname{supp}(X)| \geq C\sqrt{N/r} \cdot \sqrt{\log(2N)} \log(2r), \tag{31}$$

and

$$|X| = r \cdot |\operatorname{supp}(X)| \geq C\sqrt{Nr} \cdot \sqrt{\log(2N)} \log(2r).$$

Observe that the precondition of Lemma B.2, $|X|^2 \geq C_\delta \cdot \mu_X \cdot N$, is satisfied (assuming constant C is large enough), and hence by Lemma B.2 there exists an integer

$$1 \leq d \leq 4r\Sigma(X)/|X|^2, \tag{32}$$

such that for any integer

$$t \in [C_2 r N \Sigma(X)/|X|^2, \Sigma(X)/2] \tag{33}$$

that is a multiple of d, it holds that $t \in \mathcal{S}(X) \subseteq \mathcal{S}(A)$.

By Eq. (30) and Eq. (31), we have

$$\Sigma(X) \geq r \cdot \big(1 + 2 + \cdots + |\operatorname{supp}(X)|\big) > r \cdot |\operatorname{supp}(X)|^2/2 > 100N.$$

Take the largest subset sum $s \in \mathcal{S}(B)$ that satisfies $s \leq \Sigma(X)/2$ and s is a multiple of d, and let $S \subseteq B$ be the subset achieving $\Sigma(S) = s$. Such s exists, for example by taking $S = \varnothing$. By the assumption Eq. (27),

$$\Sigma(B) - \Sigma(S) \geq (\Sigma(A) - N) - \Sigma(X)/2 \geq (\Sigma(X) - N) - \Sigma(X)/2 > \Sigma(X)/3,$$

and hence the multiset $B \setminus S$ has size[11]

$$\begin{aligned}|B \setminus S| \geq \frac{\Sigma(B) - \Sigma(S)}{N} &> \frac{\Sigma(X)/3}{N} \\ &\geq \frac{d|X|^2}{4r} \cdot \frac{1}{3N} && \text{(by Eq. (32))} \\ &\geq d. && \text{(by } |X|^2 \geq C_\delta \cdot \mu_X \cdot N\text{)}\end{aligned}$$

So we can pick an arbitrary size-d subset $\{b_1, b_2, \ldots, b_d\} \subseteq B \setminus S$, and by the pigeonhole principle, there exist $0 \leq j < j' \leq d$ such that $b_1 + \cdots + b_j \equiv b_1 + \cdots + b_{j'} \pmod{d}$, and hence $b_{j+1} + \cdots + b_{j'}$ is a multiple of d. Denote $S' = \{b_{j+1}, \ldots, b_{j'}\} \subseteq B \setminus S$, with $|S'| \leq d$ and $\Sigma(S') \equiv 0 \pmod{d}$. Then, the subset sum $\Sigma(S \cup S') \in \mathcal{S}(B)$ is also a multiple of d. By our definition of $s = \Sigma(S)$, we must have $\Sigma(S \cup S') > \Sigma(X)/2$ due to the maximality of $\Sigma(S)$, which means

$$\Sigma(S) > \Sigma(X)/2 - \Sigma(S') \geq \Sigma(X)/2 - Nd. \tag{34}$$

Now we verify that $\Sigma(S)$ belongs to the interval defined in Eq. (33). The upper bound $\Sigma(S) \leq \Sigma(X)/2$ is guaranteed by the definition of S. For the lower bound, first note that

$$Nd \leq N \cdot 4r\Sigma(X)/|X|^2 \leq N \cdot 4r\Sigma(X)/(C_\delta rN) \leq 0.1\Sigma(X).$$

Combined with Eq. (34), this implies

$$\Sigma(S) \geq \Sigma(X)/2 - Nd \geq 0.4\Sigma(X).$$

Then, note that the lower bound of the interval Eq. (33) is

$$C_2 rN\Sigma(X)/|X|^2 \leq C_2 rN\Sigma(X)/(C_\delta rN) \leq 0.3\Sigma(X).$$

Thus, $\Sigma(S) \in [0.4\Sigma(X), 0.5\Sigma(X)]$ belongs to the interval in Eq. (33). Since $\Sigma(S)$ is a multiple of d, this implies $\Sigma(S) \in \mathcal{S}(X) \subseteq \mathcal{S}(A)$. Since $\Sigma(S) > 0$, we conclude $\Sigma(S) \in \mathcal{S}^*(A) \cap \mathcal{S}^*(B)$. □

We have proved the weaker version of Lemma 3.6. In comparison, the statement of the original Lemma 3.6 only requires $|\operatorname{supp}_r(A)| \geq C\sqrt{N/r} \cdot \sqrt{\log(2N)}$ instead of $|\operatorname{supp}_r(A)| \geq C\sqrt{N/r} \cdot \sqrt{\log(2N)}\log(2r)$ (Eq. (26)). Inspecting the proof above, we note that this extra $\log(2r)$ factor comes from the $\log(2\mu_X)$ factors in Eqs. (28) and (29) in the statement of Lemma B.2 from [BW21]. In the proof of Lemma B.2 in [BW21, Section 4.4], these $\log(2\mu_X)$ factors were incurred in the step that transforms a possibly non-uniform multiset X to a uniform multiset, where a multiset X is called *uniform* if the multiplicity $\mu_X(x)$ is the same for all elements $x \in X$; see [BW21, Lemma 4.28]. When we apply Lemma B.2 in our proof, the multiset X is already uniform (Eq. (30)), so we can avoid this transformation step in [BW21] and thus avoid the extra $\log(2\mu_X)$ factors in Eqs. (28) and (29). In this way we can prove Lemma 3.6 without the extra $\log(2r)$ factor.

B.4 Proof of Lemma 4.9

Lemma 4.9 (Deterministic balls-and-bins)**.** *Given integer r, and m sets $S_1, S_2, \ldots, S_m \subseteq [n]$ such that $|S_i| \leq r\log_2(2m)$ for all i, there is an $O(mr\log m\log r + n\log r)$-time deterministic algorithm that finds an r-coloring $C\colon [n] \to [r]$, such that for every $i \in [m]$ and every color $c \in [r]$,*

$$|\{j \in S_i : C(j) = c\}| \leq O(\log m).$$

[11] For two multisets $S \subseteq B$, the difference $B \setminus S$ is naturally defined by subtracting the multiplicities of elements.

To prove Lemma 4.9, we use the celebrated deterministic algorithm for finding a coloring of a set system that achieves small discrepancy, using the method of conditional probabilities or pessimistic estimators.

Theorem B.4 (Deterministic set balancing [Spe87, Rag88]). *Given sets $S_1, S_2, \ldots, S_m \subseteq [n]$ where $|S_i| \le b$ for all i, there is an $O(n+bm)$-time deterministic algorithm that finds $x \in \{+1,-1\}^n$, such that for every $i \in [m]$,*

$$\left|\sum_{j \in S_i} x_j\right| \le 2\sqrt{b \ln(2m)}.$$

Note that the algorithm from Theorem B.4 can run in input-sparsity time, because when coloring an element $j \in [n]$ we only need to update the pessimistic estimators for the sets S_i that contain j. To make this algorithm work in the word-RAM model with $\Theta(\log(n+m))$-bit words, the discrepancy bound is worsened by a constant factor to allow arithmetic operations with relative error $1/\operatorname{poly}(nm)$ when computing the pessimistic estimators.

Lemma 4.9 then follows from recursively applying Theorem B.4.

Proof of Lemma 4.9. Without loss of generality we assume $r \ge 2$, and assume r is a power of two, by decreasing r and increasing the size upper bound to $|S_i| \le b_0 := 2r \log_2(2m)$.

We use the following recursive algorithm with $\log_2(r)$ levels: given size-b_0 sets $S_1, \ldots, S_m$ from universe $[n]$, use Theorem B.4 to find a two-coloring $C_0 \colon [n] \to \{+1,-1\}$, and then recurse on two subproblems whose m sets are restricted to the universe $C_0^{-1}(+1)$ and the universe $C_0^{-1}(-1)$ respectively. The discrepancy of C_0 achieved by Theorem B.4 ensures these two subproblems contain sets of size at most $b_0/2 + \sqrt{b_0 \ln(2m)}$. We keep recursing in this manner, and the k-th level of the recursion tree gives a 2^k-coloring of the universe $[n]$. Finally the $\log_2(r)$-th level gives the desired r-coloring of $[n]$. By induction, the maximum size of S_i intersecting any color class is at most $b_{\log_2(r)}$, which is recursively defined as

$$b_0 = 2r \log m, \qquad b_k = b_{k-1}/2 + \sqrt{b_{k-1} \ln(2m)} \ \ (1 \le k \le \log_2(r)).$$

To solve this recurrence, first divide by b_{k-1} on both sides and get

$$\begin{aligned}
\frac{b_k}{b_{k-1}} &\le \frac{1}{2} \exp\left(2\sqrt{\ln(2m)/b_{k-1}}\right) && \text{(using } 1 + x \le e^x\text{)} \\
&\le \frac{1}{2} \exp\left(2\sqrt{2^{k-1} \ln(2m)/b_0}\right). && \text{(using } b_j \ge b_{j-1}/2 \text{ inductively)}
\end{aligned}$$

By telescoping,

$$\begin{aligned}
b_{\log_2(r)} &\le b_0 \prod_{k=1}^{\log_2(r)} \frac{1}{2} \exp\left(2\sqrt{2^{k-1} \ln(2m)/b_0}\right) \\
&= \frac{b_0}{2^{\log_2(r)}} \exp\left(2\sqrt{\ln(2m)/b_0} \sum_{k=1}^{\log_2(r)} \sqrt{2^{k-1}}\right) \\
&= \tfrac{2r \log_2(2m)}{r} \exp\left(2\sqrt{\tfrac{\ln(2m)}{2r \log_2(2m)}} \tfrac{\sqrt{r}-1}{\sqrt{2}-1}\right) \\
&= O(\log m),
\end{aligned}$$

as desired.

The total time complexity of applying Theorem B.4 at the k-th level of the recursion tree is $O(n + 2^k \cdot b_k m)$. From $b_k \geq b_{k-1}/2$, or equivalently $2^k b_k \geq 2^{k-1} b_{k-1}$, we know $2^k b_k$ is maximized at $k = \log_2(r)$. Hence, the total time complexity over all $\log_2(r)$ levels is at most

$$\begin{aligned}\sum_{k=1}^{\log_2(r)} O(n + 2^k b_k \cdot m) &\leq \log_2(r) \cdot O(n + 2^{\log_2(r)} b_{\log_2(r)} \cdot m) \\ &\leq O(n \log r + rm \log r \log m).\end{aligned}$$

□

B.5 Proof of Lemma 4.10

Lemma 4.10. *Given m sets $S_1, S_2, \ldots, S_m \in [n]$ with $|S_i| \leq b$, there is a deterministic algorithm in $O(n \log m + mb^3)$ time that computes $k \leq \log_2(2m)$ colorings $h_1, h_2, \ldots, h_k \colon [n] \to [b^2]$, such that for every $i \in [m]$ there exists an h_j that assigns distinct colors to elements of S_i.*

Proof. We iteratively apply the following claim:

Claim B.5. *Given m sets $S_1, \ldots, S_m \in [n]$ with $|S_i| \leq b$, there is a deterministic algorithm in $O(n + mb^3)$ time that computes a coloring $h \colon [n] \to [b^2]$, such that for at least $1/2$ fraction of the $i \in [m]$, h assigns distinct colors to elements of S_i.*

To prove Lemma 4.10, we use Claim B.5 to find an coloring h, and recurse on the remaining sets S_i that do not receive distinct colors under h. Finally we return all the computed colorings. Since the number of remaining sets gets halved in each iteration, we terminate within $\log_2(2m)$ iterations, and the total time complexity is $O(n \log m + mb^3)$. In the following, it remains to prove Claim B.5.

First note that a uniformly random coloring $h \colon [n] \to [b^2]$ assigns distinct colors to a fixed set S_i with probability at least $1 - \binom{|S_i|}{2} \cdot \frac{1}{b^2} \geq 1/2$ by a union bound over all $\binom{|S_i|}{2}$ pairs of distinct $x, y \in S_i$. We now use the standard method of pessimistic estimators to derandomize this random construction. Without loss of generality, assume $|S_i| = b$ for all i. We decide the colors $h(j) \in [b^2]$ sequentially for $j = 1, 2, \ldots, n$. Let $x_i^{(j)} = |S_i \cap [j]|$ denote the number of already colored elements in S_i. Assuming S_i has not received repeated colors among its first $x_i^{(j)}$ elements so far, the probability that coloring the remaining elements causes S_i to have repeated colors is at most $p(x_i^{(j)})$ given by $p(x) := \frac{1}{b^2} \cdot (x \cdot (b - x) + \binom{b-x}{2})$.

Given the already decided colors $h(1), \ldots, h(j)$, define estimator

$$e_i^{(j)} = \begin{cases} 1 & S_i \text{ has already received repeated colors,} \\ p(x_i^{(j)}) & \text{otherwise.} \end{cases}$$

Then $\sum_{i=1}^m e_i^{(0)} = m \cdot p(0) < m/2$.

Now we verify that $\mathbf{E}_{h(j+1) \in [b^2]}[e_i^{(j+1)}] \leq e_i^{(j)}$. Here it suffices to consider the case where the first $x = x_i^{(j)}$ elements of S_i have not received repeated colors, and $j + 1$ is the $(x + 1)$-st element in S_i to be colored. Then

$$\begin{aligned}\mathop{\mathbf{E}}_{h(j+1) \in [b^2]}[e_i^{(j+1)}] - e_i^{(j)} &= \left(\tfrac{x}{b^2} \cdot 1 + (1 - \tfrac{x}{b^2}) \cdot p(x+1)\right) - p(x) \\ &= -\tfrac{x(b+x)(b-x-1)}{2b^4} \\ &\leq 0.\end{aligned}$$

Hence, $\mathbf{E}_{h(j+1)\in[b^2]}[\sum_{i=1}^{m} e_i^{(j+1)}] \le \sum_{i=1}^{m} e_i^{(j)}$, and by enumerating all b^2 colors we can find a color $h(j+1) \in [b^2]$ so that $\sum_{i=1}^{m} e_i^{(j+1)} \le \sum_{i=1}^{m} e_i^{(j)}$. Eventually we can find a coloring $h\colon [n] \to [b^2]$ such that $\sum_{i=1}^{m} e_i^{(n)} \le \sum_{i=1}^{m} e_i^{(0)} < m/2$, satisfying the requirement.

Now we analyze the time complexity. In each iteration $1 \le j \le n$, we try all b^2 colors and compute the estimators. Note that the estimators $e_i^{(j)}$ are only updated when an element in S_i is colored. Hence the total time complexity for finding the coloring $h\colon [n] \to [b^2]$ is $O(n \cdot b^2 + b^2 \cdot \sum_{i=1}^{m} |S_i|) = O(nb^2 + mb^3)$, which can be further reduced to $O(n + mb^3)$ by skipping elements $j \in [n]$ that are not contained in any S_i. This finishes the proof of Claim B.5. □