Local Computation Algorithms for Knapsack: Impossibility Results, and How to Avoid Them

Clément L. Canonne
The University of Sydney, Australia

Yun Li
The University of Sydney, Australia

Seeun William Umboh
The University of Melbourne, Australia
ARC Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), Melbourne, Australia

Abstract

Local Computation Algorithms (LCA), as introduced by Rubinfeld, Tamir, Vardi, and Xie (2011), are a type of ultra-efficient algorithms which, given access to a (large) input for a given computational task, are required to provide fast query access to a consistent output solution, without maintaining a state between queries. This paradigm of computation in particular allows for hugely distributed algorithms, where independent instances of a given LCA provide consistent access to a common output solution.

The past decade has seen a significant amount of work on LCAs, by and large focusing on graph problems. In this paper, we initiate the study of Local Computation Algorithms for perhaps *the* archetypal combinatorial optimization problem, Knapsack. We first establish strong impossibility results, ruling out the existence of any non-trivial LCA for Knapsack as several of its relaxations. We then show how equipping the LCA with additional access to the Knapsack instance, namely, weighted item sampling, allows one to circumvent these impossibility results, and obtain sublinear-time and query LCAs. Our positive result draws on a connection to the recent notion of *reproducibility* for learning algorithms (Impagliazzo, Lei, Pitassi, and Sorrell, 2022), a connection we believe to be of independent interest for the design of LCAs.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Parallel algorithms; Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Packing and covering problems

Keywords and phrases Local computation algorithms, Knapsack, algorithms, lower bounds

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2025.45

Category RANDOM

Related Version *Full Version*: https://arxiv.org/abs/2504.01543

Funding *Clément L. Canonne*: Supported by the Australian Government through the Australian Research Council DE230101329.
Yun Li: Supported in part by the Australian Government through the Australian Research Council DE230101329 and DP240101353.
Seeun William Umboh: Supported by the Australian Government through the Australian Research Council DP240101353.

Acknowledgements The authors thank the anonymous reviewers for their helpful suggestions.

1 Introduction

Most classical algorithms can be divided into three phases: (1) read and pre-process the entire input; (2) perform the computations to solve the problem; and (3) finally output the solution. However, several constraints can impact the effectiveness of these algorithms in

© Clément L. Canonne, Yun Li, and Seeun William Umboh;
licensed under Creative Commons License CC-BY 4.0
Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2025).
Editors: Alina Ene and Eshan Chattopadhyay; Article No. 45; pp. 45:1–45:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

practice. For instance, computation might become infeasible with massive size input that could not fit in memory; finding an optimal solution may be exceptionally difficult while a quick response is needed; or the size of the solution itself might be too large to allow for it to be easily outputted. Different algorithmic paradigms have been introduced to address these issues, e.g., the streaming model [1] to address memory constraints, or property testing [7] to address very large inputs in the context of sublinear-time algorithms.

Local Computation Algorithms (LCAs) are sublinear time and space algorithms for search problems first introduced by [15]. The LCA model seeks to capture settings where *both* input and output are massive, so even *efficient* algorithms are too time-consuming, as both the time required to fully read the input and describe the output are too large. LCAs implement efficient query access to a small portion of a valid solution of the underlying problem without computing the whole output, while maintaining no state between the various queries it is asked to answer (for a full definition of the model, see Section 2). Due to this last requirement, LCAs are particularly well-suited for the distributed and parallel settings, as they allow for many instances of the algorithm to be run independently, each providing local query access to the same solution to the computational problem at hand.

Most of the work on LCAs, since their introduction, has focused on graph problems, such as Hypergraph Coloring, Independent Set Cover, Maximal Independent Set [15, 2, 6], and Maximum Matching [11, 3], to name a few. Yet, far fewer works have considered other computational tasks in the context of LCAs, and in particular combinatorial optimization questions. In this work, we revisit this state of affairs, focusing on one of the most fundamental combinatorial optimization problems: Knapsack. In this setting, the algorithm is provided with a (read-only) random seed, and given query access to an instance I of Knapsack: upon receiving query asking whether item i is part of an optimal solution, the algorithm is allowed to make a small (sublinear in the instance size) number of lookups[1] to I, and must answer according to some feasible solution C to Knapsack (with high probability). The algorithm must be able to answer as many such queries as desired, in arbitrary order and without maintaining a state between queries, while providing consistent access to the same solution C.

1.1 Our results and contributions

Our first set of results investigates the very possibility of obtaining an efficient local computation algorithm for Knapsack. Our first theorem rules out any sublinear-time[2] LCA who provides query access to an optimal solution:

▶ **Theorem 1** (Informal; see Theorem 13). *There is no sublinear-time LCA for Knapsack that provides consistent query access to an optimal solution.*

This hardness result, however, only rules out algorithms for the *exact* version of Knapsack: one could still hope for non-trivial approximation algorithms in the local computation model. Our second result closes this door as well:

▶ **Theorem 2** (Informal; see Theorem 14). *There is no sublinear-time LCA for Knapsack that provides consistent query access to an α-approximate solution, for any fixed $\alpha \in (0, 1]$.*

[1] To distinguish between the queries the LCA receives and the queries made by the LCA to the input Knapsack instance, we will refer the latter as "lookups."

[2] In what follows, and in particular our lower bounds, we ignore the computational complexity aspects of the algorithms and focus on their *query complexity*, that is, the worst-case number of lookups it needs to perform in order to answer an LCA query about the output. The query complexity clearly lower bounds time complexity; but this distinction is important to avoid vacuous conditional results (assuming $\mathsf{P} \neq \mathsf{NP}$), as the search version of Knapsack is NP-Hard.

One could still relax the goal in another direction: instead of requiring optimality or near-optimality of the solution provided by the LCAs, one could ask for access to any *maximal* feasible solution, regardless of its value – that is, any solution which cannot be improved by adding further items. Unfortunately, we show that even this somewhat modest goal is impossible:

▶ **Theorem 3** (Informal; see Theorem 15). *There is no sublinear-time LCA for Knapsack that provides consistent query access to a maximal feasible solution.*

These strong impossibility results, at first glance, seem to close our line of inquiry: any LCA providing access to a reasonable solution of Knapsack must essentially look up the whole input. But this relies on the fact that the algorithm has only very limited access to the instance of the problem: also random access to the instance I might seem powerful, this also does not allow it to easily leverage any feature of the instance, as the weights and profits across items are essentially independent. One avenue to circumvent these impossibility results is to equip the algorithms with a stronger and natural type of lookups to I: following [10] (in the classical setting), we consider a *weighted sampling* model, where the LCA can randomly sample items from the instance I *proportionally to their profit*. Intuitively, this should enable the algorithm to focus on the most relevant items, which are more likely to be sampled. We show that this is indeed the case, and are able to obtain a query-efficient LCA for Knapsack in this weighted sampling model:

▶ **Theorem 4** (Main theorem (Informal; see Theorem 17)). *There exists an LCA which, given weighted sampling access to the Knapsack instance, provides consistent query access to a $(1/2 - \varepsilon)$-approximation, for any fixed $\varepsilon > 0$. The LCA has query complexity*

$$(1/\varepsilon)^{O(\log^* n)} \ ,$$

where $\log^*$ *denotes the iterated logarithm.*

We elaborate on the techniques, and provide an outline of the proofs, below.

Technical overview

Our first two impossibility results, Theorems 1 and 2, follow a very similar approach: namely, we reduce the Knapsack problem on n items to computing the OR function on n bits, before invoking the known randomized query complexity lower bound for OR_n. While the resulting reduction itself is quite simple, the key aspect is to maintain the type of query access provided (i.e., simulate query access to the constructed Knapsack instance I given query access to the input $x \in \{0,1\}^n$ for the OR_n function) in a local way: that is, for the lower bound to carry through with no loss in parameters, we want to be able to answer any LCA query to I with only a small number – ideally $O(1)$ – of lookups to x.

The third impossibility result, Theorem 3, is more involved, and does not follow from a reduction. Instead, we prove it directly, first restricting ourselves to deterministic algorithms using Yao's Principle, and defining a suitable distribution of "hard Knapsack instances." These hard instances have the following structure: the weight limit is $K = 1$, and all items, except for a uniformly randomly chosen pair, have weight zero (recall that since we consider the maximal feasibility version of Knapsack in this theorem, we do not have to define or care about the items' profits). The remaining two, items i and j, have weights randomly chosen to be either $3/4, 1/4$ or $3/4, 3/4$. Note that in the first case, a maximally feasible solution must include both items i, j; while in the second case, only one of the two can – and

must – be part of the solution. The crux of the argument is then to use both the consistency and memorylessness requirements of an LCA to show that, unless it makes enough lookups to find the (randomly chosen) location of both special items i, j, any LCA must err on some sequence of (constantly many) queries, either by including both items when it should include only one, or only one when it should include both. This gives an $\Omega(n)$ lower bound for the number of lookups required to answer such an $O(1)$-length sequence of queries, and so an $\Omega(n)$ lower bound on the query complexity.

The starting point for our positive result, Theorem 4, is the work of Ito, Kiyoshima, and Yoshida [10], which provides a constant-time randomized algorithm for approximating the optimal *value* of a solution to a Knapsack instance. At a high level, given a parameter ε their algorithm works by first (implicitly) partitioning the n items, each given by a profit p_i and weight w_i, in 3 sets: (1) the set L of large items, with profit $p_i > \varepsilon^2$; (2) the set S of small items, with profit $p_i \leq \varepsilon^2$ but large "efficiency ratio" $p_i/w_i \geq \varepsilon^2$; and (3) the set G of garbage items, with low profit and low efficiency. (To interpret the notion of efficiency, recall the greedy algorithm for Fractional Knapsack, which first sorts items by non-increasing efficiency ratio before selecting as many as possible in this order.) The algorithm of [10] then samples $\tilde{O}(1/\varepsilon^4)$ items proportionally to their profit, discarding all items from G: by a coupon collector argument, with high probability all items from L will be sampled, and further the algorithm will obtain a good approximation of the frequency distribution of items in S: a set of efficiency thresholds $e_1, \ldots, e_k$, for $k = O(1/\varepsilon)$, with the guarantee that for each ℓ there is approximately an ε fraction of the total profit on items with efficiency ratio $\approx e_\ell$. Using these two facts, the algorithm can build a new *constant-size* (i.e., $O_\varepsilon(1)$-size) instance I' to Knapsack by including all large items and a suitable number of "representatives" for each efficiency from the small items, whose optimal value ε-approximates that of the original instance, and solve this new instance optimally (in exponential time in the size of the new instance) before returning its value.

To adapt this approach to the LCA setting, we must overcome several obstacles. The first is that the above algorithm does not actually solve the original instance of Knapsack (which would be a hard problem), nor an approximation of it: it solves a different, constructed instance, which just happens to have approximately the same optimal value. But our LCA needs to answer very specific queries on the *original* instance: *"is item i in the approximately optimal solution?"* While running the algorithm of Ito, Kiyoshima, and Yoshida to obtain an optimal solution to I' would let us answer this type of queries for "large" items (as these are included verbatim in the constructed instance I') and "garbage" items (as these are discarded, and not part of any solution), it does not readily let us get this information for any of the "small" items. To address this, we leverage the structure of the standard 1/2-approximation algorithm for Knapsack (which takes the best of two solutions: that returned by the greedy algorithm mentioned above, and the singleton solution obtained by including the first item left out by this greedy algorithm). By running this 1/2-approximation algorithm on the constructed instance I', we end up with a threshold τ and very simple rule to decide whether a (constructed) item i' is in the approximate solution to the (constructed) solution I': check its efficiency ratio against this threshold τ. But this threshold, by construction, also allows us (after carefully handling some corner cases) to decide whether an item i of the *original* instance is in a good feasible solution of the *original* instance I: check if it is a large item (then we can decide easily if it is in the solution), a garbage item (then it is not), and otherwise it is a small item: check its efficiency ratio against τ.

The above outline almost gets us where we want, and *would* lead (if correct) to a poly$(1/\varepsilon)$-query LCA for $(1/2 - \varepsilon)$-approximate Knapsack. Unfortunately, it has one major issue, leading to our second obstacle: the sampling step used to approximate the distribution of

efficiency profiles of the small items needs to be performed for each query, as the LCA is not allowed to maintain state between queries, *this random sampling will lead to inconsistent answers*. Indeed, even small variations in the efficiency thresholds $e_1, \ldots, e_k$ computed from this sampling step may lead to a different threshold τ, and as a result to our LCA failing to answer consistently to a single feasible solution. The solution would be to somehow have our sampling step give the same results across repetitions, which is of course not possible: one can easily design examples where randomly sampling items gives different outcomes with high probability, even across only a few repetitions; and naive attempts at rounding the sampled values suffer the same issue.

To solve this second major hurdle, we take recourse in the recent framework of *reproducible learning* (Impagliazzo, Lei, Pitassi, and Sorrell [9]), originally proposed to ensure that learning algorithms output (with high probability) the same answer when run on a fresh sample of training data. Making this connection between the LCA consistency requirement and the reproducible learning definition, we are able to leverage the *reproducible median* algorithm of [9] (suitably generalized to quantiles instead of simply median) to perform the frequency estimation step of our algorithm in a consistent fashion. Combining these building blocks together then yields our final algorithm: the $\log^* n$ dependence stemming from the use of the reproducible median, which provably requires this dependence on the domain size.

1.2 Related Work

Local Computation Algorithms

Since the introduction of Local Computation Algorithms in [15], many classical graph problems have been studied under this lens [15, 2, 6, 11, 8, 3], both from the upper and lower bound sides.

There is a strong connection between distribution algorithms and LCAs, which follows from earlier work by Parnas and Ron [14]: broadly speaking, any d-round LOCAL algorithm with degree bound Δ can be simulated by a q-query LCA, where $q = O(\Delta^d)$.

Another important design technique for LCAs (and one our algorithms leverage) is that of simulating greedy algorithms. For instance, [13] showed that some subclass of greedy algorithms for graph problems can be simulated by first randomly assigning a random number to the vertices, inducing a random ordering to the vertices. This technique was later generalized and improved by [18]. The connection to LCAs was first made explicit in [12], which used it to show how to convert some online algorithm to LCAs while preserving the same approximation ratio. This led, among others, to LCAs for maximal matching with $O(\log^3 n)$ time and space complexity and several load balancing problems with $O(\log n)$ time and space complexity.

We conclude this short review of LCAs by mentioning the recent work of Biswas, Cao, Pyne, and Rubinfeld [4], which introduced a variant of the LCA model where the input is assumed to come from some probabilistic process (e.g., random Erdős–Rényi graph). It would be interesting to see if such average-case assumptions could lead to faster LCAs for Knapsack on "almost all instances," or even allow one to bypass our impossibility results.

Knapsack

The Knapsack problem is perhaps the archetypal combinatorial optimization problem, and one of Karp's original 21 NP-Hard problems. Many variants and relaxations have been studied over the years, including under stochastic assumptions, or various restrictions on

the input. Given the vast body of work on Knapsack, we only discuss below the two most relevant to this paper, and refer the interested reader to [16, Section 3.1] for a more extensive coverage of the question and some of its approximation algorithms.

One important relaxation is when the quantity of each item to be included, instead of being binary, is allowed to take any value in $[0, 1]$. This is known as the *Fractional Knapsack*, and can be solved optimally by either linear programming or a very simple greedy algorithm which first sorts items by non-increasing ratio of p_i/w_i (profit-to-weight), and then greedily picks items in this order until the Knapsack capacity is exhausted. This greedy algorithm, while in itself does not yield any non-trivial approximation ratio for the original Knapsack problem, can be easily modified to yield a 1/2-approximation: see, e.g., [16, Exercise 3.1]. We will draw on this simple 1/2-approximation algorithm for our main algorithmic result. The second result on Knapsack we will rely on is the aforementioned work of Ito, Kiyoshima, and Yoshida [10], which, for any constant $\varepsilon > 0$, shows how to approximate (with high probability) the value of an optimal solution to a Knapsack instance to an additive $\pm\varepsilon$ in constant time (i.e., $\text{poly}(1/\varepsilon)$). Their work, which we draw inspiration from, relies on the ability to sample items from the instance with probability proportional to their profit.

2 Preliminaries

Throughout the paper, we use $[n]$ to denote the set of integers $\{1, 2 \ldots, n\}$, and standard asymptotic notation $O(\cdot), \Omega(\cdot), \Theta(\cdot)$, as well as the (slightly) less standard $\tilde{O}(\cdot)$ which omits polylogarithmic factors in its argument. We recall the definition of $\log^*$, the iterated logarithm:

$$\log^* n = \begin{cases} 0 & \text{if } n \leq 1 \\ 1 + \log^* \log n & \text{otherwise} \end{cases}$$

As out work will be concerned with approximation algorithms, we recall the general definition:

▶ **Definition 5** ((α, β)-approximation algorithm). *For any $\alpha \in [0, 1]$ and $\beta \geq 0$, an (α, β)-approximation algorithm for an optimization problem that for all instances of the problems produces a solution whose value is*

1. *at least $(\alpha \cdot \text{OPT} - \beta)$ if the problem is a maximization problem;*
2. *at most $(\alpha \cdot \text{OPT} + \beta)$ if the problem is a minimization problem*

where OPT *is the value of the optimal solution.*

We now formally define Local Computation Algorithms. This definition is the standard definition of [15], tailored to the Knapsack problem.[3]

▶ **Definition 6** (LCA for Knapsack Problem). *A $(t(n), \delta(n))$-LCA $\mathcal{A}$ for Knapsack is a (randomized) algorithm which is given access to a read-only random seed $r \in \{0, 1\}^*$, and query access to a Knapsack Instance $I = (S, K)$ on $n = |S|$ items, where the total profit of items in S is normalized to 1, and the (integer) weight of any item in S is at most K. The algorithm must support the following type of queries: on input $i \in [n]$, after making lookups to the instance I and running in time at most $t(n)$, $\mathcal{A}$ outputs whether item i is part of a*

[3] The usual definition of LCAs includes a first parameter, $s(n)$, for the space complexity of the LCA; we omit it for simplicity, as it is not the focus our of work (neither lower nor upper bounds), which is the query complexity.

feasible solution C, and must be correct with probability at least $1-\delta(n)$. Importantly, C only depends on the input instance I and random bits r used during computation. Additionally, each run has no access to previous computation results. The quantity $t(n)$ is referred to as the time complexity, and $\delta(n)$ the failure probability.[4]

We also recall two desirable properties of LCAs, which are often taken to be part of the definition:

▶ **Definition 7** (Parallelizable). *We say an LCA $\mathcal{A}$ is* parallelizable *if multiple copies of $\mathcal{A}$ can be run at the same time. Runs are consistent to the same solution as long as their input Knapsack instance I is the same and use the same random seed r during computation.*

▶ **Definition 8** (Query-Order Oblivious). *We say an LCA $\mathcal{A}$ is* query-order oblivious *if the outputs of $\mathcal{A}$ do not depend on the order of the queries but depend only on the input and the random seed r.*

In what follows, when discussing LCAs we will always refer to parallelizable, query-order oblivious LCAs. Of course, this definition by itself is not enough, as one could achieve a $(O(1), O(1), 1)$-LCA trivially by always answering "no" to any query (which would be consistent with the feasible solution $\emptyset$). We typically will require a guarantee on the *profit* of the solution as well.

We will also require the definition of *reproducible algorithm*, as introduced in [9]. A reproducible algorithm returns the same output on two distinct runs with high probability, provided that (1) the input samples it takes in both runs come from the same distribution, and (2) it uses the same internal randomness on both runs:

▶ **Definition 9** (Reproducibility, [9]). *Let D be a probability distribution over a universe $\mathcal{X}$, and let $\mathcal{A}$ be a randomized algorithm with sample access to D. An algorithm $\mathcal{A}$ is ρ-*reproducible *if*

$$\Pr_{\vec{s}_1,\vec{s}_2,r}\left[\mathcal{A}(\vec{s}_1;r) = \mathcal{A}(\vec{s}_2;r)\right] \geq 1-\rho,$$

where $\vec{s}_1$ and $\vec{s}_2$ denote sequences of samples drawn i.i.d. from D, and r denotes a random binary string representing the internal randomness used by $\mathcal{A}$.

In their work, Impagliazzo et al. provide a reproducible algorithm to compute an *approximate median*, defined as follows:

▶ **Definition 10** (τ-approximate median). *Let D be a distribution over a well-ordered domain $\mathcal{X}$. For $\tau \in [0, 1/2]$, an element $x \in \mathcal{X}$ is a τ-*approximate median *of D if $\Pr_{X\sim D}[X \leq x] \geq 1/2 - \tau$ and $\Pr_{X\sim D}[X \geq x] \geq 1/2 - \tau$.*

The guarantees of their algorithm are given in the theorem below:

▶ **Theorem 11** ([9, Theorem 4.2]). *Let $\tau, \rho \in [0, 1]$ and let $\delta = 1/3$. Let D be a distribution over $\mathcal{X}$, where $|\mathcal{X}| = 2^d$. Then there exists an algorithm $\texttt{rMedian}_{\rho,d,\tau,\delta}$ which is ρ-reproducible, outputs a τ-approximate median of D with success probability $1-\rho/2$, and has sample complexity*

$$\tilde{\Omega}\left(\left(\frac{1}{\tau^2\rho^2}\right)\cdot\left(\frac{3}{\tau^2}\right)^{\log^*|\mathcal{X}|}\right).$$

[4] Note that one would want to set $\delta(n)$ to be $1/\text{poly}(n)$, or at most $O(1/q)$ when the LCA is expected to answer q queries (so that all queries are answered successfully with high probability, by a union bound). Our lower bounds hold even for $\delta(n) = \Omega(1)$, making them even stronger.

Finally, recall that in the Knapsack problem, an instance I consists of a list of n items $(a_1, \ldots, a_n)$, each with their own non-negative value p_i and weight $w_i \geq 0$ (so that $a_i = (p_i, w_i)$), and the weight limit $K \geq 0$. The task to select a set $S \subseteq [n]$ of items which maximizes total value $\sum_{i \in S} p_i$ without exceeding the weight limit: $\sum_{i \in S} w_i \leq K$.

3 Impossibility Results

To establish our lower bounds, we will reduce from a *query complexity* problem. Recall that in query complexity, given a function $f\colon \{0,1\}^n \to \{0,1\}$, the objective is to analyse the minimum number of queries (bits) to an input $x \in \{0,1\}^n$ an algorithm has to read in order to successfully compute the value $f(x)$. The *randomised query complexity* $R(f)$ is the worst-case query complexity (over all possible inputs $x \in \{0,1\}^n$) of any randomised algorithm which correctly computes f, on any input, with probability at least $2/3$. For more on query complexity, the reader is referred to the survey [5]. In particular, we will use the following standard result on the OR function, OR_n, defined as $\mathrm{OR}_n(x) = \bigvee_{i=1}^n x_i$:

▶ **Lemma 12.** *The randomised query complexity of the OR function on n bits is $R(\mathrm{OR}_n) = \Omega(n)$.*

▶ **Theorem 13.** *Any $(t(n), 1/3)$-LCA for Knapsack which provides query access to an optimal solution must satisfy $t(n) = \Omega(n)$.*

Proof. Let $\mathcal{A}$ be any $(t(n), 1/3)$-LCA for Knapsack. We will show how to use it to compute the function OR_{n-1} with at most $t(n)$ queries to the input (and success probability at least $2/3$: by Lemma 12, this will imply $t(n) = \Omega(n)$.

On input $x \in \{0,1\}^{n-1}$, we will simulate access the Knapsack instance $I(x)$ on n items and weight limit $K = 1$ defined as follows:

$$(p_i, w_i) = \begin{cases} (x_i, 1) & \text{if } 1 \leq i \leq n-1 \\ (1/2, 1) & \text{if } i = n \end{cases}$$

That is, all n items have weight equal to the weight limit, and so any feasible solution can only contain at most one item. It is easy to see that $\mathrm{OR}_{n-1}(x) = 1$ if, and only if, an optimal solution to the Knapsack instance has value 1; otherwise, it has value $1/2$ (as $x_i = 0$ for all $1 \leq i \leq n-1$)), and the only optimal solution is the singleton $\{s_n\}$ which has profit $1/2$.

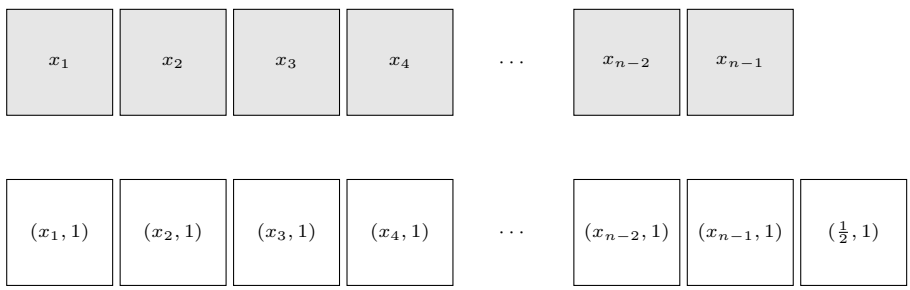

Figure 1 An illustration of the reduction. Given query access to an input $x \in \{0,1\}^{n-1}$ (top), we simulate query access to a Knapsack instance $I(x)$ (bottom) with weight limit $K = 1$, such that the n-th item belongs to the optimal solution (which is then unique) if, and only if, $\mathrm{OR}_n(x) = 0$.

Moreover, one can simulate any lookup made by $\mathcal{A}$ to this instance $I(x)$ at the cost of (at most) one query to x:

- If $\mathcal{A}$ looks up item s_n, answer $(1/2, 1)$ (no query to x needed);

- If $\mathcal{A}$ looks up item s_i for some $1 \le i \le n-1$, query x_i and answer $(x_i, 1)$ (one query to x needed).

Given the above, to compute $\mathrm{OR}_n(x)$, it suffices to make a *single* query to the LCA $\mathcal{A}$: namely, the query asking if s_n is in the optimal solution it gives access to. By the above discussion, s_n is part of an optimal solution if, and only if, $\mathrm{OR}_{n-1}(x) = 0$, so the answer given by $\mathcal{A}$ on this single query provides (with probability at least 2/3) the value $\mathrm{OR}_{n-1}(x)$.

To conclude, note since the LCA $\mathcal{A}$ can answer any single query to an optimal solution to our instance $I(x)$ (of size n) in time at most $t(n)$, it makes at most $t(n)$ lookups to the instance $I(x)$: indeed, each lookup takes unit time, so the number of lookups is a lower bound on the time complexity. Overall,

$$t(n) \ge \begin{matrix}\text{number of lookups by } \mathcal{A} \\ \text{to the instance } I(x)\end{matrix} \ge \begin{matrix}\text{number of queries} \\ \text{made to the input } x\end{matrix}$$

So our reduction allows us, on any input x, to compute (with probability at least 2/3) $\mathrm{OR}_{n-1}(x)$ with at most $t(n)$ queries to x: by Lemma 12, it follows that $t(n) = \Omega(n)$. ◀

The above result rules out LCAs which provide local access to an optimal Knapsack solution. We will reduce from the same query complexity problem to a slightly different Knapsack Instance to show that any LCA that gives query access to *any* finite approximate Knapsack solution takes at least $\Omega(n)$ time.

▶ **Theorem 14.** *Fix any $\alpha \in (0, 1]$. Any $(t(n), 1/3)$-LCA for Knapsack which provides query access to an α-approximate feasible solution must satisfy $t(n) = \Omega(n)$.*

Proof. Let $\mathcal{A}$ be any $(t(n), 1/3)$-LCA for α-approximation Knapsack, and fix any $0 < \beta < \alpha$. On input $x \in \{0,1\}^{n-1}$, we simulate access the Knapsack Instance $I(x)$ on n items and weight limit $K = 1$ defined as follows:

$$(p_i, w_i) = \begin{cases} (x_i, 1) & \text{if } 1 \le i \le n-1 \\ (\beta, 1) & \text{if } i = n \end{cases}$$

i.e., the same instance as in the proof of Theorem 13, but with the profit of the last item set to β. As before, any feasible solution of $I(x)$ contains at most one item as all n items have the weight equal to the weight limit. The value of the optimal solution is then either 1 (if at least one of the the x_i's is 1) or β (if all x_i's are 0, in which case $\{s_n\}$ is the optimal solution. Thus $\{s_n\}$ is an α-approximation solution if, and only if, $\mathrm{OR}_{n-1}(x) = 0$; in this case, $s_n = (\beta, 1)$ has higher profit than any other items which makes it the optimal solution, hence it is also a unique α-approximation solution for $I(x)$. Otherwise the optimal solution contains an item with profit 1, and $\{s_n\}$ is not an α-approximation solution as $\beta < 1 \cdot \alpha$. Again, we can simulate any lookup by $\mathcal{A}$ to this instance $I(x)$ at the cost of one query to x:

- If $\mathcal{A}$ looks up item s_n, answer $(\beta, 1)$ (no query to x needed);
- If $\mathcal{A}$ looks up item s_i for some $1 \le i \le n-1$, query x_i and answer $(x_i, 1)$ (one query to x needed).

It is sufficient to compute $\mathrm{OR}_{n-1}(x)$ by making a single query to the LCA $\mathcal{A}$: by asking whether s_n is in a α-approximation solution it gives access to. Since the LCA $\mathcal{A}$ can answer any single query to a α-approximation solution to our instance $I(x)$ (of size n) in at most $t(n)$ time, $\mathcal{A}$ makes at most $t(n)$ lookups to the instance $I(x)$. And any lookup made to $I(x)$ can by simulated by at most one query to x. Therefore, $t(n)$ is lower bounded by the query complexity of $\mathrm{OR}_{n-1}(x)$. By Lemma 12, it follows that $t(n) = \Omega(n)$. ◀

Since we can choose α to be arbitrarily close to 0, we get that there is no sublinear time LCA that answers to any finite approximation Knapsack solution with failure probability at most 1/3.

Having ruled out in Theorems 13 and 14 any sublinear-time LCAs for optimal and approximate Knapsack, we further relax the constraint to show that even asking for an different relaxation of Knapsack is impossible. That is, instead of giving answers consistent to a finite approximation solution, we ask if an LCA can give access to a *maximal feasible* Knapsack solution in sublinear time. Instead of proving the impossibility result directly for LCAs, we will show that no *deterministic* algorithm can provide query access to a maximal feasible solution on some difficult input distribution in sublinear time. By Yao's Principle [17], this will imply that no randomized algorithm can provide sublinear time query access to maximal feasible Knapsack.

▶ **Theorem 15.** *Any $(t(n), 4/5)$-LCA for Knapsack which provides query access to an maximal feasible solution must satisfy $t(n) = \Omega(n)$.*

Proof. We define the distribution of items S of the input Knapsack instance I as follows:

1. select uniformly at random a pair of indices $(i, j) \in \{1, 2, \ldots, n\}^2$
2. assign $w_i = 3/4$, and

$$w_j = \begin{cases} 1/4 & \text{with probability } 1/2 \\ 3/4 & \text{with probability } 1/2 \end{cases}$$

3. every other item $k \notin \{i, j\}$ is assigned $w_k = 0$.

Let $\mathcal{A}$ be a *deterministic* algorithm for Maximal Feasible Knapsack with time complexity $t(n) < \frac{n}{11}$ and probability of success at least 4/5.

The weight limit K is 1. (Since we are looking at maximal feasible solutions, the profits of the items are irrelevant: we set them all to 0 (i.e., set $p_k = 0$ for all k) and ignore them for the rest of the proof.) Note that if $w_j = 1/4$, then the (unique) maximal solution is $C = \{s_1, \ldots, s_n\}$ (all items); while if $w_j = 3/4$, then the two distinct maximal solutions are $C^{(-i)} = \{s_1, \ldots, s_n\} \setminus \{s_i\}$ and $C^{(-j)} = \{s_1, \ldots, s_n\} \setminus \{s_j\}$. The following claim will be crucial:

▶ **Lemma 16.** *Suppose $\mathcal{A}$ receives query s_k, such that $w_k = 3/4$. Then with probability at least 9/10 over the choice of the instance I, $\mathcal{A}$ outputs* yes *(i.e., that this item is in the maximal feasible solution).*

Proof. Given such a query s_k, in view of our choice of input distribution, there are two options: either $k = i$ (since $w_i = 3/4$ always), or $k = j$ and $w_j = 3/4$. So $k \in \{i, j\}$: let k' be the other value of the couple (that if, if $k = i$ then $k' = j$, and vice-versa).

On query s_k, we assume that the algorithm knows that $w_k = 3/4$ (since this only take one lookup to the instance I to learn this, given k). The algorithm makes deterministically a sequence of at most $q \leq t(n) < \frac{n}{11}$ lookups to the instance I, where lookup ℓ reveals the weight w_ℓ of item ℓ. Without loss of generality, we can assume that the algorithm does not look up an item it already knows the weight of; so it looks up at most $q < \frac{n}{11}$ new items. Since k' is uniformly distributed among the remaining $n - 1$ items, the probability that any of these q lookups (even made adaptively) finds the other non-zero-weight item $s_{k'}$ is at most

$$\frac{n}{n-1} \cdot \frac{1}{11} \leq \frac{1}{10}$$

since the answer to every single lookup ℓ made is answered by "$w_\ell = 0$." We denote by E_i the event

$$E_k = \{\text{on query } k,\ A \text{ does not look up any other item with non-zero weight }\}$$

By the above, $\Pr[E_k] \geq \frac{9}{10}$. Now, the claim is that conditioned on E_k, the algorithm must say yes to the query (i.e., that s_k is in the maximal solution it provides access to). To see why, note that given only the information given (namely, the fact that the k-th item has weight 3/4, and that all the other items looked up have weight 0), the three following cases are possible given our distribution over instances I:

- the k-th item is s_j, and there is somewhere another item, s_i, with weight 3/4 (this has probability 1/3)
- the k-th item is s_i, and there is somewhere another item, s_j, with weight 3/4 (this has probability 1/3)
- the k-th item is s_i, and there is somewhere another item, s_j, with weight 1/4 (this has probability 1/3)

In the first two cases, the algorithm could reply either yes or no and still be consistent with one of the two maximal solutions; but in the third case, the algorithm must respond yes since *all* items are in the (unique) maximal solution. Since we are in the third case with probability 1/3 (and that the algorithm gets no useful information except with probability at most 1/10), to achieve error less than $1/5 < 1/3 - 1/10$ overall the algorithm must respond yes. ◀

By the *principle of deferred decisions*, we can make the random choice of deciding the value of w_j (either 1/4 or 3/4 uniformly at random) when $\mathcal{A}$ looks up s_j.

Consider what happens when the LCA $\mathcal{A}$ receives queries for item s_i, then s_j. By the above claim (for $k = i$), on query s_i, $\mathcal{A}$ will output yes with probability at least 9/10 (over the choice of the instance).

On the *second* query s_j, we then have two choices: with probability 1/2, $w_j = 1/4$ and the algorithm can safely respond yes. With probability 1/2, however, $w_j = 3/4$, and by the same claim (for $k = j$) will then output yes with probability at least 9/10.

This means, by a union bound, that the algorithm is correct with probability at most

$$\Pr[\bar{E}_i] + \Pr[w_j \neq \frac{3}{4}] + \Pr[\bar{E}_j] \leq \frac{1}{10} + \frac{1}{2} + \frac{1}{10} < \frac{4}{5}$$

since with probability $\Pr[E_i \cap \{w_j = 3/4\} \cap E_j]$ it says yes to item s_i with weight 3/4 *and* item s_j which has weight 3/4; i.e., the solution it is consistent with is not feasible.

This implies that any deterministic algorithm for maximal feasible Knapsack which is correct with probability at least 4/5 on this specific input distribution must make at least $\frac{n}{11}$ lookups. By Yao's Principle, this implies that any randomized algorithm cannot answer Maximal Feasible Knapsack with success probability 4/5 unless $t(n) \geq \frac{n}{11}$. ◀

4 Algorithmic Result: an LCA given Weighted Sampling Access

We have shown in the last section that it is impossible to obtain a (non-trivial) LCA for Knapsack, or even some relaxations of it. The crux of the issue lies in the "needle in a haystack" phenomenon: in order to be consistent across queries, an LCA seemingly needs to get a global sense of the distribution of weights and profits of the instance, and in particular the largest profits. Yet, given only query access to the instance, finding these requires a very large number of lookups.

To mitigate this, we adapt the weighted sampling model considered in [10], which equips the algorithm with the ability to sample items with probability proportional to their profit (assuming, essentially without loss of generality, that the total profit and weight are both normalized to 1). In this setting, we establish the following result:

▶ **Theorem 17.** *Algorithm 2 (*$\texttt{LCA}-\texttt{KP}$*) is a* $(t(n), \varepsilon)$*-LCA for Knapsack which provides query access to an* $(1/2, 6\varepsilon)$*-approximate solution, where* $t(n) = (1/\varepsilon)^{O(\log^* n)}$.

The remainder of this section is dedicated to proving this theorem. Before we describe the general idea of the algorithm, we first cover the weighted sampling model and results from [10] we will use in our algorithm. Fixing $\varepsilon \in (0, 1]$, the items of the Knapsack instance $I = (S, K)$ are partitioned into three sets:

$$
\begin{aligned}
L(I) &:= \{(p, w) \in S : p > \varepsilon^2\} && \text{(items with high profit)}\\
S(I) &:= \{(p, w) \in S : p \le \varepsilon^2 \text{ and } p/w \ge \varepsilon^2\} && \text{(items with low profit but high efficiency)}\\
G(I) &:= \{(p, w) \in S : p \le \varepsilon^2 \text{ and } p/w < \varepsilon^2\} && \text{(items with low profit and low efficiency)}
\end{aligned}
$$

The items in $L(I)$, $S(I)$, $G(I)$ are referred as *large items*, *small items* and *garbage items* respectively. The following lemma, which follows from a coupon-collector type argument, ensures that one can sample all sufficiently heavy elements with high probability.

▶ **Lemma 18** (Lemma 2, [10]). *Let* $B = \{(w, p) \in I : p \ge \delta\}$. *By taking* $\lceil 6\delta^{-1}(\log \delta^{-1} + 1)\rceil$ *samples using weighted sampling, all items of* B *are sampled at least once with probability at least* $5/6$.

Importantly, this success probability can be easily amplified by repetition from $5/6$ to $1 - \varepsilon$ at the cost of a logarithmic factor in $1/\varepsilon$, which we will rely on later.

Small items are partitioned over intervals of efficiency such that the total profits of items within each interval is approximately ε. The set of small items in each efficiency intervals is denoted by $A_0(I), A_1(I), ..., A_t(I)$, and the corresponding *efficiency thresholds* are denoted by a non-increasing efficiency sequence $e_1 \ge e_2 \ge ... \ge e_t$, such that:

$$
\begin{aligned}
A_0(I) &:= \{(p, w) \in S(I) : p/w \ge e_1\}\\
A_k(I) &:= \{(p, w) \in S(I) : e_k > p/w \ge e_{k+1}\} && (1 \le k \le t-1)\\
A_t(I) &:= \{(p, w) \in S(I) : e_t > p/w\}
\end{aligned}
$$

A crucial definition is then that of an *equally partitioning sequence*, which is a sequence of efficiency threshold such that the total profit of each efficiency interval satisfies $\sum_{i \in A_k(I)} p_i \approx \varepsilon$:

▶ **Definition 19** (Equally Partitioning Sequence (EPS), [10]). *We say the efficiency sequence* $e_1, e_2, ..., e_t$ *is* equally partitioning with respect to I *if, for all* $0 \le i \le t-1$, $A_i(I)$ *has total profit within* $[\varepsilon, \varepsilon + \varepsilon^2)$ *and* $A_t(I)$ *has total profit within* $[0, \varepsilon + \varepsilon^2)$.

The authors of [10] also show how to obtain an EPS, with probability at least $1 - O(1/m^2)$, from sampling $O(\frac{1}{\varepsilon^4} \log \frac{1}{\varepsilon})$ items.[5] Now, their algorithm, given weighted sampling access to the original instance I and a parameter ε, constructs a different instance $\tilde{I}$ (of size $\text{poly}(1/\varepsilon)$) as follows:

[5] The statement in [10] only claims success probability 5/6; however, inspection of the proof show the authors establish the stronger claim.

1. It samples a multiset of $O(\frac{1}{\varepsilon^4}\log\frac{1}{\varepsilon})$ items, and puts all the large items into a set M. (By Lemma 18, all items of $L(I)$ are included in M, except with probability 1/6.)
2. It samples a multiset of $O(\frac{1}{\varepsilon^4}\log\frac{1}{\varepsilon})$ items, and uses it to obtain an Equally Partitioning Sequence $\tilde{e}_1,\ldots,\tilde{e}_t$, except with probability 1/6.
3. From the above set M and the EPS, it constructs a new instance $\tilde{I}=(\tilde{S},\tilde{K})$ as follows. It creates three disjoint sets $L(\tilde{I}),S(\tilde{I}),G(\tilde{I})$ mimicking the sets $L(I),S(I),G(I)$ defined for I:

$$L(\tilde{I}) := M,$$
$$S(\tilde{I}) := \bigcup_{k=0}^{t-1} A_k(\tilde{I}),\ \text{where } A_k(\tilde{I}) \text{ contains exactly } \lfloor\varepsilon^{-1}\rfloor \text{ copies of } (\varepsilon^2,\varepsilon^2/\tilde{e}_{k+1}),$$
$$G(\tilde{I}) := \emptyset$$

and defines $\tilde{S} := L(\tilde{I})\cup S(\tilde{I})\cup G(\tilde{I})$ and $\tilde{K}=K$.

Note that $t=O(1/\varepsilon)$ and $|M|\le|L(I)|\le 1/\varepsilon^2$, and so $\tilde{I}$ is an instance on only $O(1/\varepsilon^2)$ items. We call the above procedure the *$\tilde{I}$-construction algorithm*. The central lemma they show then relates the value of the optimal solution to the new Knapsack instance $\tilde{I}$ to that of the original instance I.

▶ **Lemma 20** (Lemma 1, [10]). *Let $I=(S,K)$ be an instance of Knapsack, and let $\tilde{I}$ be the instance constructed from I using an equally partitioning sequence with respect to I. Then, $\mathrm{OPT}(\tilde{I})-\varepsilon$ is a $(1,6\varepsilon)$-approximation to $\mathrm{OPT}(I)$.*

4.1 General Idea of the Algorithm

As in the algorithm of [10], we will use weighted sampling to gather items with high profit and learn the distribution of the efficiencies of small items with high probability. Then we will compute the greedy solution of the corresponding simplified Knapsack instance $\tilde{I}$, as defined above. (Recall that the greedy algorithm for the Knapsack problem first sorts the items over their efficiency, then it includes the items in descending order until the output set reaches the weight limit; we will refer to the efficiency of the first item that the greedy solution cannot fully include as the *efficiency cut-off*.) The greedy solution will either be the set of items with efficiency higher than the efficiency cut-off, or the one item with efficiency cut-off if that item's profit is higher than the total profit of the items with higher efficiency.

Our LCA will determine the queried item is chosen or not based on the efficiency cut-off of the greedy solution of the simplified Knapsack of that run. As discussed in Section 1.1, one challenge lies in the fact that the simplified instance $\tilde{I}$ is constructed by sampling, hence items of the simplified instance and the corresponding efficiency cut-off may (and likely will) be different between each run, jeopardizing the consistency requirement of the LCA. To address this issue, we generalize the *reproducible median* algorithm from [9] to a *reproducible quantile* algorithm that will output the exact same approximation quantiles of the efficiency of the items with high probability. We will start with the *reproducible quantile* algorithm.

4.2 Reproducible Quantile Algorithm

We will leverage the reproducible algorithm of [9] (Theorem 11) to compute the quantiles in a reproducible fashion. The idea is to reduce the task of computing quantiles to that of computing the median. Given an array T of n elements, to find the p-quantile of T, we can add x many $-\infty$ and y many $+\infty$ to T, such that $x+pn=(1-p)n+y$ and $x+y=n$, let the new array be T'. The median of array T' will be the same value as the p-quantile of array T.

Similarly, we can compute the reproducible p-quantile of a given distribution D by computing the reproducible median of a new distribution D' such that:

1. The domain of D' is the union of the domain of D (assumed to be of size 2^d) and $\{+\infty, -\infty\}$; note that we can bound domain size of D' by 2^{d+1},
2. For any element e from distribution D, $\Pr_{i\sim D'}[i = e] = \frac{\Pr_{i\sim D}[i=e]}{2}$,
3. $\Pr_{i\sim D'}[i = -\infty] = x, \Pr_{i\sim D'}[i = +\infty] = y, x = (1-p)/2, y = p/2$.

Algorithm 1 `rQuantile`$(\vec{s}, p)$.

Require: $\vec{s}$: a sample of n_d elements drawn i.i.d. from D'; p: queried percentile in D
Require: Parameters:
- ρ: target reproducibility parameter
- d: specifies domain size $|\mathcal{X}| = 2^d$
- τ: target accuracy of p-quantile
- β: target failure probability
- n_d: sample complexity of `rMedian` to output a ρ-reproducible τ-approximation median with failure probability β on domain $|\mathcal{X}| = 2^d$

Ensure: returns a τ-approximate p-quantile of D

1: Run `rMedian`$(\vec{s})$ with reproducibility parameter ρ, domain size 2^{d+1}, target accuracy $\tau/2$ and target failure probability β and return the result v

▶ **Theorem 21** (`rQuantile` Correctness and Query Complexity). *Let $\tau, \beta, \rho \in [0, 1]$. Let D be a distribution over $\mathcal{X}$, with $|\mathcal{X}| = 2^d$. Then* `rQuantile`$_{\rho,d,\tau,\beta}$ *(Algorithm 1) is a reproducible algorithm with sample complexity*

$$\tilde{O}\left(\left(\frac{1}{\tau^2(\rho-\beta)^2}\right)\cdot\left(\frac{12}{\tau^2}\right)^{\log^*|\mathcal{X}|+1}\right)$$

which, on input p, outputs a τ-approximate p-quantile of D with probability at least $1-\beta$.

Proof. We first argue the correctness of `rQuantile`. `rQuantile` returns a value v which is a ρ-reproducible $\tau/2$-approximation median of distribution D' except with probability β. Thus,

$$\Pr_{i\sim D'}[i \le v] \ge 1/2 - \tau/2 \text{ and } \Pr_{i\sim D'}[i \ge v] \ge 1/2 - \tau/2.$$

By the definition of D',

$$\Pr_{i\sim D}[i \le v] = 2\Pr_{i\sim D'}[i \le v \text{ and } i \ne -\infty] \ge 2(1/2 - \tau/2 - (1-p)/2) = p - \tau$$

and

$$\Pr_{i\sim D}[i \ge v] = 2\Pr_{i\sim D'}[i \ge v \text{ and } i \ne +\infty] \ge 2(1/2 - \tau/2 - p/2) = 1 - p - \tau.$$

Therefore, `rQuantile` returns a τ-approximation p-quantile of distribution D. By [9, Theorem 4.8], `rMedian`$_{\rho,d,\tau,\beta}$ has sample complexity of

$$\tilde{O}\left(\left(\frac{1}{\tau^2(\rho-\beta)^2}\right)\cdot\left(\frac{3}{\tau^2}\right)^{\log^*|\mathcal{X}|}\right).$$

Since `rQuantile` runs `rMedian` with target accuracy $\tau/2$ and domain size 2^{d+1}, the sample complexity of `rQuantile` is

$$\tilde{O}\left(\left(\frac{1}{\tau^2(\rho-\beta)^2}\right)\cdot\left(\frac{12}{\tau^2}\right)^{\log^*|\mathcal{X}|+1}\right),$$

as desired. ◀

Mapping to a finite domain

`rQuantile` can only compute quantiles with respect to a distribution D over a *finite* domain $\mathcal{X}$ (of some size of the form 2^d) since the recursion of `rMedian` terminates when the domain size become 2. However, in our case the distribution D is the distribution of efficiencies, which (since both profits and weights are *a priori* arbitrary positive numbers) have domain $\mathbb{R}_{>0}$. To address this, we first note that the weights are not quite arbitrary: before the normalization assumption, they were all positive integers. Assuming a $\mathrm{poly}(n)$ bound on the number of bits needed to describe any given weight, we get that the domain of the (normalized) weights is of the form $\mathcal{W} = \{1/B, 2/B, \ldots, 1\}$, where $B = 1/2^{\mathrm{poly}(n)}$.

We can assume the same for the profits, i.e., that all profits take value in the domain $\mathcal{P} = \{0, 1/B', 2/B', \ldots, 1\}$ for some $B' = 1/2^{\mathrm{poly}(n)}$.[6] Overall, this ensures that each efficiency is of the form

$$\frac{B}{B'}\cdot\frac{a}{b}$$

for some $0 \le a \le B'$ and $1 \le b \le B$. This leads to the domain $\mathcal{X}$ of the efficiencies being known and finite (but huge), of size bounded by $|\mathcal{X}| = (B'+1)B = 2^{\mathrm{poly}(n)}$. Note that this implies that the dependence on the domain size from `rQuantile`, which involves $\log^*|\mathcal{X}|$ (in the exponent), will be sublinear in n, as

$$\log^*|\mathcal{X}| = \log^*(2^{\mathrm{poly}(n)}) = O(\log^* n).$$

4.3 From Reproducible Quantiles to our Knapsack LCA

We now put all the pieces together to analyze our final algorithm, Algorithm 2, thus establishing Theorem 17. We will show that the solution C the algorithm answers according to is (1) feasible (Lemma 23), (2) a good approximate solution (Lemma 24), and (3) consistent across queries (Lemma 25); before bounding the sample (query) complexity in Lemma 26.

Without loss of generality, we will refer to the items by their index after sorting in $\mathtt{CONVERT-GREEDY}$. We separate $\mathtt{CONVERT-GREEDY}$ and $\mathtt{MAPPING-GREEDY}$ as we will only call $\mathtt{CONVERT-GREEDY}$ in $\mathtt{LCA-KP}$. We start with the below lemma, whose proof is deferred to the full version due to space constraints.

▶ **Lemma 22.** *Conditioned on the sample $\vec{R}$ containing all elements of $L(I)$ after Algorithm 2, then, with probability at least $1 - 13\varepsilon/36$, the approximate quantile sequence $\tilde{e}_1, \ldots, \tilde{e}_{t'}$ in* $\mathtt{LCA-KP}$ *(Algorithm 2) is an EPS with respect to I.*

[6] Another possibility is to avoid this bit complexity argument and instead use a rounding procedure, standard in the fully polynomial-time approximation scheme (FPTAS) for Knapsack (see [16, Section 3.2]), which rounds each profit "on-the-fly" to a multiple of ε/n, as the price of an $1-\varepsilon$ approximation factor loss in the solution. This would bound the size of the domain of the profits by n/ε, for any fixed choice of $\varepsilon > 0$. We here describe the bit-complexity bound argument, as it mimics that of the weights.

Algorithm 2 $\texttt{LCA}-\texttt{KP}$.

Require: $I = (S, K)$: Knapsack Instance, where S is the item set and K is the weight limit; i: queried index of item in S

1: Let $\vec{R}$ be a sample of size $m := O(\frac{1}{\varepsilon^4}\log\frac{1}{\varepsilon})$ from S ▷ Succeeds with probability $1-\varepsilon/3$ by Lemma 18
2: Remove all items with profit less or equal than ε^2 and duplicate items from $\vec{R}$
3: $L(\tilde{I}) \leftarrow \vec{R}$
4: **if** $1 - p(L(\tilde{I})) \geq \varepsilon$ **then**
5: Set $q := \frac{\varepsilon+\varepsilon^2/2}{1-p(L(\tilde{I}))}$, $t = \lfloor q^{-1} \rfloor, \tau = \varepsilon^2/5, \rho = \frac{\varepsilon^2}{18}, \beta = \rho/2, n_{rq}$ = sample complexity of $\texttt{rQuantile}_{\rho,d,\tau,\beta}$
6: Let $\vec{Q}$ be a sample of size $a := \lceil \frac{3n_{rq}}{2(1-p(L(\tilde{I})))} \rceil$ from I
7: Remove every item with profit greater than ε^2 from $\vec{Q}$
8: $\vec{E} \leftarrow \{p/w \mid (p, w) \in \vec{Q}\}$
9: **for** $k = 1, 2, \ldots, t$ **do**
10: $\tilde{e}_k \leftarrow \texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E}, 1 - kq)$
11: **if** $\tilde{e}_t < \varepsilon^2$ **then**
12: $t' \leftarrow t - 1$
13: **else**
14: $t' \leftarrow t$
15: EPS $\leftarrow (\tilde{e}_1, \tilde{e}_2, \ldots, \tilde{e}_{t'})$
16: **else**
17: EPS $\leftarrow \emptyset$
18: Construct Knapsack Instance $\tilde{I} = (\tilde{S}, K)$ based on $L(\tilde{I})$ and efficiency sequence EPS as in step 3 of the $\tilde{I}$-construction algorithm
19: $\text{Index}_{\text{large}}, e_{\text{small}}, B_{\text{indicator}} \leftarrow \texttt{CONVERT}-\texttt{GREEDY}(\tilde{I}, \text{EPS})$
20: **if** $p_i > \varepsilon^2$ **then**
21: return yes if $i \in \text{Index}_{\text{large}}$, otherwise return no
22: **else if** $e_{\text{small}} \neq -1$ and $p_i/w_i \geq e_{\text{small}}$ **then**
23: return yes
24: return no

Algorithm 3 $\texttt{CONVERT}-\texttt{GREEDY}$.

Require: $\tilde{I}$ and $\tilde{e}_1, \tilde{e}_2, \ldots, \tilde{e}_t$, the Equally Partitioning Sequence that $\tilde{I}$ is constructed from

1: Let $\{(p_1, w_1), \ldots, (p_{|\tilde{S}|}, w_{|\tilde{S}|})\}$ be the items in $\tilde{S}$ sorted by their efficiency in non-increasing order.
2: Let $1 \leq j \leq |\tilde{S}|$ be the largest index such that $\sum_{i=1}^{j} w_i \leq K$
3: Let $1 \leq k \leq t$ be the largest index such that $\tilde{e}_k > p_j/w_j$
4: **if** $j = |\tilde{S}|$ or $\sum_{i=1}^{j} p_i \geq p_{j+1}$ **then**
5: $\text{Index}_{\text{large}} \leftarrow \{\text{the index in } I \text{ corresponding to } i \mid 1 \leq i \leq j, (p_i, w_i) \in I, p_i > \varepsilon^2\}$
6: **if** $k \geq 3$ **then**
7: $e_{\text{small}} \leftarrow e_{k-2}$
8: **else**
9: $e_{\text{small}} \leftarrow -1$
10: $B_{\text{indicator}} \leftarrow$ false
11: **else**
12: $\text{Index}_{\text{large}} \leftarrow \{\text{the index in } I \text{ corresponding to } j + 1\}$
13: $e_{\text{small}} \leftarrow -1, B_{\text{indicator}} \leftarrow$ true
14: **return** $\text{Index}_{\text{large}}, e_{\text{small}}, B_{\text{indicator}}$

Algorithm 4 MAPPING − GREEDY.

Require: $\text{Index}_{\text{large}}, e_{\text{small}}, B_{\text{indicator}}$: Output of CONVERT − GREEDY; I: the original Knapsack Instance

1: $C \leftarrow \{(p_i, w_i) \mid i \in \text{Index}_{\text{large}}\}$
2: **if** $B_{\text{indicator}} = \text{false}$ and $e_{\text{small}} \neq -1$ **then**
3: $\quad C \leftarrow C \cup \{(p, w) \mid (p, w) \in S(I) \text{ and } p/w \geq e_{\text{small}}\}$
4: **return** C

▶ **Lemma 23.** *The output C of* MAPPING − GREEDY *(Algorithm 4) is a feasible solution of I.*

Proof. We argue the feasibility of C, that is $w(C) \leq K$. First we consider the special case where $C = \{(p_{j+1}, w_{j+1})\}$. In this case $p_{j+1} > \sum_{i=1}^{j} p_i$, since all items in $S(\tilde{I})$ have the same profit ε^2, (p_{j+1}, w_{j+1}) must be an item with profits greater than ε^2, which implies $(p_{j+1}, w_{j+1}) \in L(\tilde{I})$. Since $L(\tilde{I}) \subseteq L(I)$ and all items in I have weight at most K, $w(C) = w_{j+1} \leq K$ and so C is feasible.

Now we consider the general case where

$$C = \{(p, w) \in L(I) \mid p/w \geq \ p_j/w_j\} \cup \{(p, w) \in S(I) \mid p/w \geq \tilde{e}_{k-2}\}.$$

Let $A_0(I), A_1(I), \ldots, A_t(I)$ be the partition of items in $S(I)$ with respect to efficiency sequence $\tilde{e}_1, \tilde{e}_2, \ldots, \tilde{e}_t$. Let $A_0(\tilde{I}), A_1(\tilde{I}), \ldots, A_{t-1}(\tilde{I})$ be the partition of items in $S(\tilde{I})$ with respect to efficiency sequence $\tilde{e}_1, \tilde{e}_2, \ldots, \tilde{e}_t$. By construction, $A_i(\tilde{I})$ contains $\lfloor \varepsilon^{-1} \rfloor$ copies of item $(\varepsilon^2, \varepsilon^2/\tilde{e}_{i+1})$ for $i = 0, 1, \ldots, t-1$. We consider the total weight of the items in $S(\tilde{I})$ that are included in the greedy solution $\texttt{GREEDY}(\tilde{I})$. Since $\tilde{e}_k > p_j/w_j$, all items in $A_i(\tilde{I})$ must be included in the greedy solution $\texttt{GREEDY}(\tilde{I})$, for $i = 0, 1, \ldots, k-1$. Therefore,

$$\begin{aligned}
w(S(\tilde{I}) \cap \texttt{GREEDY}(\tilde{I})) &\geq \sum_{i=0}^{k-1} w(A_i(\tilde{I})) \\
&= \lfloor \varepsilon^{-1} \rfloor \cdot \varepsilon^2/\tilde{e}_1 + \lfloor \varepsilon^{-1} \rfloor \cdot \varepsilon^2/\tilde{e}_2 + \cdots + \lfloor \varepsilon^{-1} \rfloor \cdot \varepsilon^2/\tilde{e}_k \\
&= \lfloor \varepsilon^{-1} \rfloor \varepsilon^2 \sum_{i=1}^{k} (1/\tilde{e}_i) \\
&\geq (\varepsilon^{-1} - 1)\varepsilon^2 \sum_{i=1}^{k} (1/\tilde{e}_i) \\
&= (\varepsilon + \varepsilon^2) \sum_{i=1}^{k-2} (1/\tilde{e}_i) - 2\varepsilon^2 \sum_{i=1}^{k-2} (1/\tilde{e}_i) + (\varepsilon - \varepsilon^2)(1/\tilde{e}_{k-1} + 1/\tilde{e}_k) \\
&> (\varepsilon + \varepsilon^2) \sum_{i=1}^{k-2} (1/\tilde{e}_i) - 2\varepsilon^2 \sum_{i=1}^{k-2} (1/\tilde{e}_{k-2}) + (\varepsilon - \varepsilon^2)(1/\tilde{e}_{k-2} + 1/\tilde{e}_{k-2}) \\
&= (\varepsilon + \varepsilon^2) \sum_{i=1}^{k-2} (1/\tilde{e}_i) - 2\varepsilon^2 (k - 1 - \varepsilon^{-1})/\tilde{e}_{k-2}.
\end{aligned}$$

Since the total profit of the items in I is 1 and $p(A_i(I)) \geq \varepsilon$, the length of the efficiency sequence t is at most $\lfloor \varepsilon^{-1} \rfloor$. Since $k \leq t$ and $k < \lfloor \varepsilon^{-1} \rfloor \leq \varepsilon^{-1}$, combining both inequalities give us $(k - 1 - \varepsilon^{-1}) < 0$. Therefore,

$$w(S(\tilde{I}) \cap \texttt{GREEDY}(\tilde{I})) \geq (\varepsilon + \varepsilon^2) \sum_{i=1}^{k-2} (1/\tilde{e}_i) = \sum_{i=1}^{k-2} ((\varepsilon + \varepsilon^2)/\tilde{e}_i) \geq \sum_{i=0}^{k-3} w(A_i(I)).$$

The last inequality is implied by the fact that $\tilde{e}_{i+1}$ lower bounds the efficiency of the items in $A_i(I)$ while the total profit of $A_i(I)$ is upper bounded by $\varepsilon + \varepsilon^2$. Now we consider the total weight of items in $L(\tilde{I})$ that are included in $\texttt{GREEDY}(\tilde{I})$. Since $L(\tilde{I}) \subseteq L(I)$, we get $w(L(\tilde{I}) \cap \texttt{GREEDY}(\tilde{I})) = w(L(I) \cap \texttt{GREEDY}(\tilde{I}))$. Combining these inequalities, we have

$$\begin{aligned} w(C) &= w(\{(p,w) \in L(I) \mid p/w \geq \ p_j/w_j\}) + w(\{(p,w) \in S(I) \mid p/w \geq \tilde{e}_{k-2}\}) \\ &\leq w(L(\tilde{I}) \cap \texttt{GREEDY}(\tilde{I})) + w((S(\tilde{I}) \cap \texttt{GREEDY}(\tilde{I})) \\ &= w(\texttt{GREEDY}(\tilde{I})) \\ &\leq \tilde{K} = K\,, \end{aligned}$$

concluding the proof. ◀

▶ **Lemma 24.** *If $\tilde{I}$ contains all large items from I, then C is a $(1/2, 6\varepsilon)$-approximation solution of I.*

Proof. We consider the difference between the total profit of C and $\texttt{GREEDY}(\tilde{I})$. Let $\mathrm{OPT}(I)$ be the optimal solution of instance I and $\mathrm{OPT}(\tilde{I})$ be the optimal solution of instance $\tilde{I}$. We first consider the case where $C = \{(p_{j+1}, w_{j+1})\}$. In this case $p(C) = p(\texttt{GREEDY}(\tilde{I}))$. By Lemma 20,

$$\begin{aligned} p(C) &= p(\texttt{GREEDY}(\tilde{I})) \geq 1/2p(\mathrm{OPT}(\tilde{I})) \geq 1/2(p(\mathrm{OPT}(I)) - 5\varepsilon) \\ &= 1/2p(\mathrm{OPT}(I)) - 5\varepsilon/2. \end{aligned}$$

Now we consider the case where $C = \{(p,w) \in L(I) \mid p/w \geq \ p_j/w_j\} \cup \{(p,w) \in S(I) \mid p/w \geq \tilde{e}_{k-2}\}$. Since we assume $\tilde{I}$ contains all *large items* from I, therefore $\{(p,w) \in L(I) \mid p/w \geq \ p_j/w_j\} = \texttt{GREEDY}(\tilde{I}) \cap L(\tilde{I})$. We consider the difference between the profit of $\texttt{GREEDY}(\tilde{I}) \cap S(\tilde{I})$ and $\{(p,w) \in S(I) \mid p/w \geq \tilde{e}_{k-2}\}$. The least efficient item (p_j, w_j) is either in $S(\tilde{I})$ or $L(\tilde{I})$.

Subcase 1. If $(p_j, w_j) \in S(\tilde{I})$, then $\bigcup_{i=0}^{i=k-1} A_i(\tilde{I}) \subset \texttt{GREEDY}(\tilde{I}) \cap S(\tilde{I}) \subseteq \bigcup_{i=0}^{i=k} A_i(\tilde{I})$. Since $p(A_i(I)) \in [\varepsilon, \varepsilon + \varepsilon^2)$ and $p(A_i(\tilde{I})) \leq \varepsilon$ for $i = 0, 1, \ldots, t-1$,

$$\begin{aligned} p(\{(p,w) \in S(I) \mid p/w \geq \tilde{e}_{k-2}\}) &= \sum_{i=0}^{k-3} p(A_i(I)) \geq \sum_{i=0}^{k-3} p(A_i(\tilde{I})) \geq \sum_{i=0}^{k} p(A_i(\tilde{I})) - 3\varepsilon \\ &\geq p(\texttt{GREEDY}(\tilde{I}) \cap S(\tilde{I})) - 3\varepsilon. \end{aligned}$$

Subcase 2. If $(p_j, w_j) \in L(\tilde{I})$, then $\bigcup_{i=0}^{i=k-1} A_i(\tilde{I}) = \texttt{GREEDY}(\tilde{I}) \cap S(\tilde{I})$. Since $p(A_i(I)) \in [\varepsilon, \varepsilon + \varepsilon^2)$ and $p(A_i(\tilde{I})) \leq \varepsilon$ for $i = 0, 1, \ldots, t-1$,

$$\begin{aligned} p(\{(p,w) \in S(I) \mid p/w \geq \tilde{e}_{k-2}\}) &= \sum_{i=0}^{k-3} p(A_i(I)) \geq \sum_{i=0}^{k-3} p(A_i(\tilde{I})) \geq \sum_{i=0}^{k-1} p(A_i(\tilde{I})) - 2\varepsilon \\ &= p(\texttt{GREEDY}(\tilde{I}) \cap S(\tilde{I})) - 2\varepsilon. \end{aligned}$$

In both cases, $p(\{(p,w) \in S(I) \mid p/w \geq \tilde{e}_{k-2}\}) \geq p(\texttt{GREEDY}(\tilde{I}) \cap S(\tilde{I})) - 3\varepsilon$. Using the assumption that $\tilde{I}$ contains $L(I)$, we obtain the following lower bound on $p(C)$:

$$\begin{aligned} p(C) &= p(\{(p,w) \in L(I) \mid p/w \geq \ p_j/w_j\}) + p(\{(p,w) \in S(I) \mid p/w \geq \tilde{e}_{k-2}\}) \\ &\geq p(\texttt{GREEDY}(\tilde{I}) \cap S(\tilde{I})) + p(\texttt{GREEDY}(\tilde{I}) \cap L(\tilde{I})) - 3\varepsilon \\ &= p(\texttt{GREEDY}(\tilde{I})) - 3\varepsilon \\ &\geq 1/2(p(\mathrm{OPT}(\tilde{I}))) - 3\varepsilon \\ &\geq 1/2(p(\mathrm{OPT}(I)) - 5\varepsilon) - 3\varepsilon \\ &\geq 1/2p(\mathrm{OPT}(I)) - 6\varepsilon. \end{aligned}$$

Therefore, C is a $(1/2, 6\varepsilon)$-approximation solution of I. ◀

▶ **Lemma 25** (Consistency). $\texttt{LCA}-\texttt{KP}$ *provides query access consistent to a* $(1/2, 6\varepsilon)$*-approximation solution of* I *with probability* $1-\varepsilon$.

Proof. The output of $\texttt{LCA}-\texttt{KP}$ is determined by $\tilde{I}$ only. Thus, as long as two runs of $\texttt{LCA}-\texttt{KP}$ constructs the same $\tilde{I}$, their answer for any index i will be the same. Every run satisfying the following conditions constructs the same instance $\tilde{I}$:

1. all items in $L(I)$ are included in $\vec{R}$,
2. $|\vec{E}|$ is at least the sample complexity of $\texttt{rQuantile}$,
3. the approximate quantiles returned by $\texttt{rQuantile}$ are both reproducible and within error τ.

We consider the failure probability of Condition (1). By Lemma 18, all items in $L(I)$ can be gathered by taking $m = O(\frac{1}{\varepsilon^4}\log\frac{1}{\varepsilon})$ samples with probability $1-\varepsilon/3$. Therefore, the failure probability of Condition (1) is at most $\varepsilon/3$. Conditioned on success of (1), he failure probability of Condition (2) is at most $\varepsilon/3$ as proved in Lemma 22.

Conditioned on the success of (2), we consider Condition (3). Fix k and internal randomness r, let $\vec{E_1}, \vec{E_2}$ be two independent samples given to $\texttt{rQuantile}_{\rho,d,\tau,\beta}$ on two distinct runs of $\texttt{LCA}-\texttt{KP}$. By the definition of reproducibility in [9],

$$\Pr_{\vec{E_1},\vec{E_2},r}[\texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E_1},k) = \texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E_2},k)] \geq 1-\rho.$$

Let D_k be the set of all possible output of $\texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E},k)$. Note that D_k is finite since it shares the same domain with $\texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E},k)$. Since $\vec{E_1}$ and $\vec{E_2}$ are two independent samples draw from $S(I)\cup G(I)$, we have:

$$\begin{aligned}&\Pr_{\vec{E_1},\vec{E_2},r}[\texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E_1},k) = \texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E_2},k)]\\ =&\sum_{x\in D_k}\left(\Pr_{\vec{E}\sim S(I)\cup G(I)}[\texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E},k)=x]\right)^2.\end{aligned}$$

Let $e_k \in D_k$ such that for any $x \in D_k$,

$$\Pr_{\vec{E}\sim S(I)\cup G(I)}[\texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E},k)=e_k] \geq \Pr_{\vec{E}\sim S(I)\cup G(I)}[\texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E},k)=x].$$

Let $y = \Pr_{\vec{E}\sim S(I)\cup G(I)}[\texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E},k)=e_k]$, since $\sum_{x\in D_k}\left(\Pr_{\vec{E}\sim S(I)\cup G(I)}[\texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E},k)=x]\right) = 1$. Therefore,

$$y^2\cdot 1/y \geq \sum_{x\in D_k}\left(\Pr_{\vec{E}\sim S(I)\cup G(I)}[\texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E},k)=x]\right)^2 = 1-\rho,$$

and so $\Pr_{\vec{E},r}[\texttt{rQuantile}_{\rho,d,\tau,\beta}(\vec{E},k) = e_k] \geq 1-\rho$. The probability that e_k is not a τ-approximation quantile is at most β. By union bound $\texttt{rQuantile}_{\rho,d,\tau,\beta}$ outputs some value e_k and a τ-approximation $(1-kq)$-quantile, except with probability $\rho+\beta$. Moreover, $\texttt{rQuantile}_{\rho,d,\tau,\beta}$ is called t times where $t = \lfloor q^{-1}\rfloor \leq q^{-1} = \frac{1-p(L(I))}{\varepsilon+\varepsilon^2/2} \leq \varepsilon^{-1}$. By union bound, all calls to $\texttt{rQuantile}$ are consistent to some τ-approximation quantile sequence $e_1, e_2, \ldots, e_{t'}$ except with probability $\varepsilon^{-1}(\rho+\beta) = \varepsilon/12$.

By a union bound, all three conditions succeed except with probability $\varepsilon/3+\varepsilon/3+\varepsilon/12 < \varepsilon$. By Lemma 24, $\texttt{LCA}-\texttt{KP}$ gives query access consistent to a $(1/2, 6\varepsilon)$-approximation solution of I with probability $1-\varepsilon$. ◀

▶ **Lemma 26** (Sample Complexity). *The sample complexity of* $\texttt{LCA}-\texttt{KP}$ *is* $(1/\varepsilon)^{O(\log^* n)}$.

Proof. $\texttt{LCA}-\texttt{KP}$ requires two samples to compute: $\vec{Q}$ and $\vec{R}$. On Line 4, the algorithm checks if $1-p(L(\tilde{I}))\geq\varepsilon$, so $|\vec{Q}|\leq\lceil\frac{3n_{rq}}{2\varepsilon}\rceil$. As a result,

$$\begin{aligned}|\vec{R}|+|\vec{Q}| &= O\left(\frac{1}{\varepsilon^4}\log\frac{1}{\varepsilon}\right)+\frac{3}{2\varepsilon}\cdot\tilde{O}\left(\left(\frac{1}{\tau^2(\rho-\beta)^2}\right)\cdot\left(\frac{12}{\tau^2}\right)^{\log^*|\mathcal{X}|+1}\right)\\ &= \tilde{O}\left(\varepsilon^{-7}\cdot\left(\varepsilon^{-6}\right)^{O(\log^* n)}\right)\\ &= (1/\varepsilon)^{O(\log^* n)},\end{aligned}$$

as desired. ◀

Combining Lemma 25 and Lemma 26 establishes Theorem 17.

5 Discussion and Future Work

The obvious question left open by our work is whether the $\exp(O(\log^* n))$ dependence on the instance size can be improved, or removed altogether. Doing so would require an entirely different approach to deal with the consistency issue, as the reproducible median algorithm does require a dependence (albeit very mild) on the domain size. Another parameter of interest is the probability of failure of the LCA, here set to $\delta=\varepsilon$: ideally, one would keep δ as a free parameter, with a runtime growing as $\text{polylog}(1/\delta)$: unfortunately, the use of replicable algorithms appears to require a polynomial dependence in $1/\delta$, so achieving this would require new ideas and techniques. Relatedly, it would be interesting to improve the approximation ratio to obtain an $1\pm\varepsilon$ approximation, for arbitrary constant $\varepsilon>0$.

The connection we made between LCAs and to reproducible algorithms was key to our main algorithmic result. We believe that exploring further this interplay, and how insights and techniques from reproducible learning algorithms can be used to design new or better LCAs for a variety of tasks, would be an fruitful direction of study.

Another, orthogonal direction of research would be to see if allowing a very small amount of memory (i.e., polylogarithmic in the input size) between computations, as allowed in the original definition of LCAs of [15], would enable significantly better algorithms for Knapsack – or even bypass our impossibility results entirely. If so, it would be interesting to characterize the minimum amount of local memory sufficient for this purpose.

Finally, as mentioned in the introduction, it would be interesting to see if the relaxation to *average-case* local computation, as introduced in [4], would lead to either faster LCAs for Knapsack with weighted sampling access, or help circumvent our impossibility results absent this type of access.

References

1. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29. ACM, 1996. doi:10.1145/237814.237823.
2. Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *SODA*, pages 1132–1139. SIAM, 2012. doi:10.1137/1.9781611973099.89.
3. Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Local computation algorithms for maximum matching: New lower bounds. In *FOCS*, pages 2322–2335. IEEE, 2023. doi:10.1109/FOCS57990.2023.00143.
4. Amartya Shankha Biswas, Ruidi Cao, Edward Pyne, and Ronitt Rubinfeld. Average-case local computation algorithms. *CoRR*, abs/2403.00129, 2024. doi:10.48550/arXiv.2403.00129.

5 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.

6 Mohsen Ghaffari. Local computation of maximal independent set. In *FOCS*, pages 438–449. IEEE, 2022. doi:10.1109/FOCS54457.2022.00049.

7 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. In *FOCS*, pages 339–348. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548493.

8 Christoph Grunau, Slobodan Mitrovic, Ronitt Rubinfeld, and Ali Vakilian. Improved local computation algorithm for set cover via sparsification. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2993–3011. SIAM, 2020. doi:10.1137/1.9781611975994.181.

9 Russell Impagliazzo, Rex Lei, Toniann Pitassi, and Jessica Sorrell. Reproducibility in learning. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 818–831, 2022. doi:10.1145/3519935.3519973.

10 Hiro Ito, Susumu Kiyoshima, and Yuichi Yoshida. Constant-time approximation algorithms for the knapsack problem. In *Theory and Applications of Models of Computation: 9th Annual Conference, TAMC 2012, Beijing, China, May 16-21, 2012. Proceedings 9*, pages 131–142. Springer, 2012. doi:10.1007/978-3-642-29952-0_17.

11 Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. *Algorithmica*, 77(4):971–994, 2017. doi:10.1007/S00453-016-0126-Y.

12 Yishay Mansour, Aviad Rubinstein, Shai Vardi, and Ning Xie. Converting online algorithms to local computation algorithms. In *Automata, Languages, and Programming: 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I 39*, pages 653–664. Springer, 2012. doi:10.1007/978-3-642-31594-7_55.

13 Huy N Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 327–336. IEEE, 2008. doi:10.1109/FOCS.2008.81.

14 Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007. doi:10.1016/J.TCS.2007.04.040.

15 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 223–238. Tsinghua University Press, 2011. URL: http://conference.iiis.tsinghua.edu.cn/ICS2011/content/papers/36.html.

16 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

17 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 222–227, 1977. doi:10.1109/SFCS.1977.24.

18 Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. Improved constant-time approximation algorithms for maximum matchings and other optimization problems. *SIAM Journal on Computing*, 41(4):1074–1093, 2012. doi:10.1137/110828691.