Author/s:
Canonne, C;Li, Y;Umboh, SW

Title:
Brief Announcement: Local Computation Algorithms for Knapsack: impossibility results, and how to avoid them

Date:
2025-06-13

Citation:
Canonne, C., Li, Y. & Umboh, S. W. (2025). Brief Announcement: Local Computation Algorithms for Knapsack: impossibility results, and how to avoid them. Proceedings of the Annual ACM Symposium on Principles of Distributed Computing, Part of F216205, pp.372-374. ACM. https://doi.org/10.1145/3732772.3733515.

Persistent Link:
https://hdl.handle.net/11343/362037

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

Clément L. Canonne
The University of Sydney
Sydney, Australia
clement.canonne@sydney.edu.au

Yun Li
The University of Sydney
Sydney, Australia
yunli97@outlook.com

Seeun William Umboh
The University of Melbourne,
ARC OPTIMA
Melbourne, Australia
william.umboh@unimelb.edu.au

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. We study the Knapsack Problem under LCA model. 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.

CCS Concepts

• **Theory of computation** → *Distributed algorithms*; *Parallel algorithms*; **Streaming, sublinear and near linear time algorithms**; **Packing and covering problems**.

Keywords

Local computation algorithms, Knapsack, algorithms, lower bounds

ACM Reference Format:
Clément L. Canonne, Yun Li, and Seeun William Umboh. 2025. Brief Announcement: Local Computation Algorithms for Knapsack: Impossibility Results, and How to Avoid Them. In *ACM Symposium on Principles of Distributed Computing (PODC '25), June 16–20, 2025, Huatulco, Mexico.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3732772.3733515

1 Introduction

Local Computation Algorithms (LCAs) are sublinear time and space algorithms for search problems first introduced by [9]. 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. 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 [1, 5, 9], and Maximum Matching [2, 8], 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 queries to I, and must answer according to some feasible solution $S = S(I)$ 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 S.

2 Preliminaries

We first recall the definitions of approximation algorithms and local computation algorithms, as well as some key notions our work relies upon.

Definition 2.1 ((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 $(\alpha \cdot \mathrm{OPT} - \beta)$ for a maximization problem;
(2) at most $(\alpha \cdot \mathrm{OPT} + \beta)$ for a minimization problem;

where OPT is the value of the optimal solution.

*Full version available at [4].

PODC '25, Huatulco, Mexico

ACM ISBN 979-8-4007-1885-4/25/06
https://doi.org/10.1145/3732772.3733515

We now formally define Local Computation Algorithms. This definition is the standard definition of [9], tailored to the Knapsack problem.[1]

Definition 2.2 (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 queries to the instance I and running in time at most $t(n)$, $\mathcal{A}$ outputs whether item i is part of a 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.[2]

We will also require the definition of *reproducible algorithm*, as introduced in [6]. 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 2.3 (Reproducibility, [6]). 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

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

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 2.4 (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 $\Pr_{X \sim D}[X \geq x] \geq 1/2 - \tau$.

3 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[3] LCA who provides query access to an optimal solution:

Theorem 3.1. *Any $(t(n), 1/3)$-LCA for Knapsack which provides query access to an optimal solution must satisfy $t(n) = \Omega(n)$.*

[1]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.

[2]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.

[3]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 queries to the input 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 P ≠ NP), as the search version of Knapsack is NP-Hard.

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 3.2. *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 $t(n) = \Omega(n)$.*

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

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 query 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 query 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 query access to I: following [7] (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 3.4 (Main theorem). *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 for our positive result, below.

The starting point for our positive result, Theorem 3.4, is the work of Ito, Kiyoshima, and Yoshida [7], 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 [7] 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 $\text{poly}(1/\varepsilon)$-query LCA for $(2+\varepsilon)$-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 [6]), 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 [6] (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.

4 Discussion and Future Work

The obvious question left open 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.

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, or more generally for distributed computation problems, would be an fruitful direction of study.

Finally, it would be interesting to see if the relaxation to *average-case* local computation, as introduced in [3], 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, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. 2012. Space-efficient local computation algorithms. In *SODA*. SIAM, 1132–1139.

[2] Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. 2023. Local Computation Algorithms for Maximum Matching: New Lower Bounds. In *FOCS*. IEEE, 2322–2335.

[3] Amartya Shankha Biswas, Ruidi Cao, Edward Pyne, and Ronitt Rubinfeld. 2024. Average-Case Local Computation Algorithms. *CoRR* abs/2403.00129 (2024).

[4] Clément L. Canonne, Yun Li, and Seeun William Umboh. 2025. Local Computation Algorithms for Knapsack: impossibility results, and how to avoid them. arXiv:2504.01543 [cs.DS] https://arxiv.org/abs/2504.01543 Full version of this paper.

[5] Mohsen Ghaffari. 2022. Local Computation of Maximal Independent Set. In *FOCS*. IEEE, 438–449.

[6] Russell Impagliazzo, Rex Lei, Toniann Pitassi, and Jessica Sorrell. 2022. Reproducibility in learning. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. 818–831.

[7] Hiro Ito, Susumu Kiyoshima, and Yuichi Yoshida. 2012. 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*. Springer, 131–142.

[8] Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. 2017. Local Computation Algorithms for Graphs of Non-constant Degrees. *Algorithmica* 77, 4 (2017), 971–994.

[9] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. 2011. Fast Local Computation Algorithms. In *Innovations in Computer Science - ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, Bernard Chazelle (Ed.). Tsinghua University Press, 223–238.