DOCTORAL THESIS

Algorithms for Knapsacks, Paths and Strings

Alejandro Cassis

A dissertation submitted towards
the degree *Doctor of Natural Sciences* of the
Faculty of Mathematics and Computer Science
of Saarland University

Saarbrücken, 2024

Colloqium

Date	17/09/2024
Dean of the Faculty	Univ. Prof. Dr. Roland Speicher
Chair of the Committee	Prof. Dr. Kurt Mehlhorn
Reporters	Prof. Dr. Karl Bringmann
	Prof. Dr. Danupon Nanongkai
	Prof. Dr. Aviad Rubinstein
Academic Assistant	Dr. Tomasz Kociumaka

The knowing self is partial in all its guises, never finished, whole, simply there and original; it is always constructed and stitched together imperfectly, and *therefore* able to join with another, to see together without claiming to be another.

Donna Haraway
Situated Knowledges

Pertenecer es re-habitar. Negar el origen abstracto o puro del universo y abrazar su materialidad: eso es pertenecer. La primera habitación, por lo tanto, es la huella.

Cristina Rivera Garza
Autobiografía del algodón

Abstract

In this thesis we study three problems:

1. *Knapsack:* The Knapsack problem is a classic combinatorial optimization problem. We give a collection of improved exact and approximation algorithms for Knapsack and some of its variants. Our study is guided by the connection between Knapsack and min-plus convolution, a central problem in fine-grained complexity.

2. *Sublinear Edit Distance:* The edit distance is a popular and practically motivated measure of similarity between texts. We focus on *sublinear-time* algorithms, which aim to approximate the edit distance k of two texts without reading them entirely. Our main result is a $k^{o(1)}$-approximation in time $O(n/k + k^{2+o(1)})$. This constitutes a quadratic improvement over the previous state of the art, and matches an unconditional lower bound for small k (up to subpolynomial factors in the running time and approximation factor).

3. *Negative Weight Single-Source Shortest Paths:* Computing shortest paths from a source in a weighted directed graph is a fundamental problem. When all edge weights are nonnegative, the classic Dijkstra's algorithm solves this problem in near-linear time. It has been a long-standing open question to obtain a near-linear time algorithm when the graph contains negative edge weights. This has been solved recently in a breakthrough by Bernstein, Nanongkai and Wulff-Nilsen, who presented an algorithm in time $O(m \log^8 n \log W)$. Our contribution is an improvement by nearly 6 log factors.

Zusammenfassung

In dieser Doktorarbeit untersuchen wir drei Probleme:

1. *Knapsack:* Knapsack ist ein klassisches kombinatorisches Optimierungsproblem. Wir präsentieren eine Sammlung von verbesserten exakten und approximativen Algorithmen für Knapsack und einige seiner Varianten. Unsere Studie wird geleitet von der Verbindung zwischen Knapsack und der Min-Plus-Faltung, einem zentralen Problem der feinkörnigen Komplexität.

2. *Sublineare Editierdistanz:* Die Editierdistanz ist ein beliebtes und praktisch motiviertes Maß für die Ähnlichkeit zwischen Texten. Wir konzentrieren uns auf Algorithmen mit *sublinearer* Zeit, die darauf abzielen, die Editierdistanz k von zwei Texten zu approximieren, ohne sie vollständig zu lesen. Unser Hauptergebnis ist eine $k^{o(1)}$-Approximation in der Zeit $O(n/k+k^{2+o(1)})$. Dies stellt eine quadratische Verbesserung gegenüber dem bisherigen Stand der Technik dar und entspricht einer unbedingten unteren Schranke für kleine k (bis auf Subpolynomialfaktoren in der Laufzeit und im Approximationsfaktor).

3. *Negativ gewichtete kürzeste Pfade von einer Quelle:* Die Berechnung der kürzesten Pfade von einer Quelle in einem gewichteten gerichteten Graphen ist ein grundlegendes Problem. Wenn alle Kantengewichte nicht-negativ sind, löst der klassische Dijkstra-Algorithmus dieses Problem in nahezu linearer Zeit. Es ist seit langem eine offene Frage, einen Algorithmus mit nahezu linearer Zeit zu erhalten wenn der Graph negative Kantengewichte enthält. Diese Frage wurde kürzlich in einem Durchbruch von Bernstein, Nanongkai und Wulff-Nilsen beantwortet, mit einen Algorithmus in der Zeit $O(m \log^8 n \log W)$. Unser Beitrag ist eine Verbesserung um fast 6 log-Faktoren.

Acknowledgements

I want to thank my advisor Karl Bringmann for his support and guidance throughout these years. It was not easy to start a PhD during a global pandemic, and I could not imagine better conditions than what he generously offered. I feel honored to have worked with him and have learned so much in this process. I also want to thank all of my co-authors, without whom this work would not be possible. Special thanks to Nick Fischer from whom I learned a lot, specially the joy of doing research with a friend.

I am grateful and indebted to all my friends who helped and cared for me in multiple ways throughout this unfinished and neverending journey. You show me and remind me of the lights at every breath.

Las palabras no me alcanzan para expresar la gratitud a mis padres y hermana por su amor y apoyo incondicional. Todo mi trabajo está dedicado a ustedes. Gracias a mis abuelos y antepasados por permitirme cada paso y en particular, este logro. Espero honrarlos con mis acciones.

Finally, I am deeply grateful to my partner and companion, Monika. Thank you for all your boundless love, encouragement and support. It is all to be with, and to love.

Preface

This thesis is divided into three independent parts. The first part focuses on algorithms for various knapsack problems. The second on sublinear-time algorithms for computing the edit distance. The third and final part is devoted to the negative weight single source shortest paths problem in graphs.

During my PhD I have worked on a range of different problems My list of publications at the time of submitting this thesis can be found below.

[BC22] Karl Bringmann and Alejandro Cassis. “Faster Knapsack Algorithms via Bounded Monotone Min-Plus-Convolution.” In: *ICALP*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 31:1–31:21. DOI: 10.4230/LIPIcs.ICALP.2022.31.

[BC23] Karl Bringmann and Alejandro Cassis. “Faster 0-1-Knapsack via Near-Convex Min-Plus-Convolution.” In: *ESA*. Vol. 274. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 24:1–24:16. DOI: 10.4230/LIPIcs.ESA.2023.24.

[BCF23] Karl Bringmann, Alejandro Cassis, and Nick Fischer. “Negative-Weight Single-Source Shortest Paths in Near-Linear Time: Now Faster!” In: *FOCS*. IEEE, 2023, pp. 515–538. DOI: 10.1109/FOCS57990.2023.00038.

[Bri+21] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Marvin Künnemann. “Fine-Grained Completeness for Optimization in P.” In: *APPROX-RANDOM*. Vol. 207. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 9:1–9:22. DOI: 10.4230/LIPIcs.APPROX/RANDOM.2021.9.

[Bri+22a] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Marvin Künnemann. “A Structural Investigation of the Approximability of Polynomial-Time Problems.” In: *ICALP*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 30:1–30:20. DOI: 10.4230/LIPIcs.ICALP.2022.30.

[Bri+22b] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. “Almost-optimal sublinear-time edit distance in the low distance regime.” In: *STOC*. ACM, 2022, pp. 1102–1115. DOI: 10.1145/3519935.3519990.

[Bri+22c] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. “Improved Sublinear-Time Edit Distance for Preprocessed Strings.” In: *ICALP*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 32:1–32:20. DOI: 10.4230/LIPIcs.ICALP.2022.32.

[Bri+24] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Tomasz Kociumaka. "Faster Sublinear-Time Edit Distance." In: *SODA*. SIAM, 2024, pp. 3274–3301. DOI: 10.1137/1.9781611977912.117.

[CKW23] Alejandro Cassis, Tomasz Kociumaka, and Philip Wellnitz. "Optimal Algorithms for Bounded Weighted Edit Distance." In: *FOCS*. IEEE, 2023, pp. 2177–2187. DOI: 10.1109/FOCS57990.2023.00135.

These publications can be categorized as follows. In [Bri+21] and [Bri+22a] we established a fine-grained classification of a large class of optimization problems, these two papers were obtained as extensions to my Master's thesis. In [BC22; BC23] we studied various knapsack problems in different settings, these two publications form the core of Part I. In [Bri+22b; Bri+22c; Bri+24; CKW23] we studied edit distance in various settings. Two of these papers, [Bri+22b] and [Bri+24] focus on sublinear-time algorithms for edit distance, which are the focus of Part II. Finally, in [BCF23] we studied the negative single source shortest paths problem, which constitutes Part III.

Contents

Part I

Knapsack

1 Knapsack and MinPlus Convolution

This part of the thesis is based on the publications [BC22; BC23]. I contributed an equal share of the work, and more than half of the write-up.

[BC22]	Karl Bringmann and Alejandro Cassis. "Faster Knapsack Algorithms via Bounded Monotone Min-Plus-Convolution." In: *ICALP*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 31:1–31:21. DOI: 10.4230/LIPIcs.ICALP.2022.31.
[BC23]	Karl Bringmann and Alejandro Cassis. "Faster 0-1-Knapsack via Near-Convex Min-Plus-Convolution." In: *ESA*. Vol. 274. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 24:1–24:16. DOI: 10.4230/LIPIcs.ESA.2023.24.

An integer linear program (ILP) is an optimization problem that can be compactly expressed as computing

$$\max\{\, c^T x \mid Ax = b, x \in \mathbb{Z}^n_{\geq 0}, x \leq u \,\},$$

where the input consists of $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m, c \in \mathbb{Z}^n$ and $u \in \mathbb{Z}^n_{\geq 0}$. ILPs can model a large variety of problems, and have been extensively studied both in theory and practice.

In this part of the thesis we study one of the simplest types of ILPs, the *Knapsack* problem. Here, we are given a set $\mathcal{I}$ of n items with weights $w_1, \dots, w_n \in \mathbb{N}$ and profits $p_1, \dots, p_n \in \mathbb{N}$, along with a knapsack capacity $W \in \mathbb{N}$. The goal is to find a subset of the items with total weight at most W that maximizes the total profit. This can be expressed as computing

$$\mathrm{OPT} := \max\left\{ \sum_{i=1}^n p_i x_i \mid \sum_{i=1}^n w_i x_i \leq W, x \in \{\, 0, 1 \,\}^n \right\}.$$

That is, Knapsack is an ILP with n variables and one constraint.

Knapsack is a classical and fundamental problem lying in the intersection of computer science and operations research, and has been studied for decades. To showcase its centrality, we remark that (i) Knapsack is among Karp's 21 NP-hard problems [Kar72], (ii) Bellman gave an algorithm for it running in pseudopolynomial time

$O(n \cdot \min\{W, \mathrm{OPT}\})$ as a prime example of dynamic programming [Bel57] (which has become a *textbook* algorithm) and (iii) it was among the first problems to have a *fully polynomial-time approximation scheme* (FPTAS) [IK75]. We refer the reader to [KPP04] for an entire book on the topic and its variants.

Despite the ubiquity and long history of Knapsack, it is still intensely studied, and several questions remain open about its complexity. At a birds' eye, our work in this thesis aims to find *best possible* algorithms for Knapsack problems.

An Inspiring Story: Subset Sum In the Subset Sum problem, we are given a set of n integers $A = \{a_1, \dots, a_n\}$, and a target integer t. The task is to decide if there is a subset of A whose sum equals t. This can be modeled as deciding the feasibility of the following ILP with one constraint $\{\sum_i a_i x_i = t \mid x \in \{0,1\}^n\}$.

In this sense, we can think of Subset Sum as one-dimensional version of Knapsack: in the former we decide feasibility of an ILP with one constraint, in the latter we additionally have an objective function that we want to maximize.

Similarly as for Knapsack, a textbook dynamic programming algorithm solves Subset Sum in time $O(nt)$. In a breakthrough result, Koiliaris and Xu improved this running time to $\widetilde{O}(\sqrt{n}t)$ [KX19][1]. This constituted the first polynomial-time improvement over the dynamic programming algorithm in about 60 years. Shortly after, Bringmann further improved the running time to $\widetilde{O}(n+t)$ [Bri17] (later improved by polylogarithmic factors by Jin and Wu [JW19]). Remarkably, this running time has a matching lower bound (up to subpolynomial factors) under both the Set Cover Hypothesis [Cyg+16] and the Strong Exponential Time Hypothesis [Abb+22b]. In that sense, the $\widetilde{O}(n+t)$-time algorithm is *best possible.*

This yields a clean understanding of the complexity of pseudopolynomial-time algorithms for Integer Linear Programming in arguably its simplest form (up to subpolynomial factors). Moreover, this line of research has sparked the study of Subset Sum in multiple new directions, like output sensitive algorithms e.g. [BN20], conditionally optimal approximation schemes e.g. [BN21] and conditionally optimal algorithms for variants of the problem [Axi+21; CI21; Kle22; DMZ23]. Along the way, this research developed a rich algorithmic toolkit (notably, the *partition and convolve* paradigm that has been used in many recent algorithms for Subset Sum, Knapsack and related problems see e.g. [Bri17; Bat+18; BC22; Bri+22d; JR23; Cha18; KX19; DJM23; BN21; KPR23]) as well as conditional lower bound technology [Abb+22b] that has been successfully applied to other problems [Abb+22a; HMS22; JR23].

As mentioned earlier, Subset Sum can be seen as the simplest type of Integer Linear Programming: it consists in checking feasibility of an ILP with one constraint. *Its success story inspires our work to develop an analogous understanding of the "next step" in ILPs, namely Knapsack.* The hope is to develop algorithmic tools that are applicable to other problems. More ambitiously, this can be seen as a stepping stone towards an analogous

1. In this chapter, we use $\widetilde{O}(\cdot)$ to suppress polylogarithmic factors in the input size and the largest input number.

understanding of general ILPs (i.e. with an arbitrary number of constraints).

Our Focus: Knapsack The aforementioned conditional lower bounds for Subset Sum carry over to Knapsack. In particular, there is no algorithm for Knapsack in time $W^{1-\delta}2^{o(n)}$ for any $\delta > 0$ under the Strong Exponential Time Hypothesis and the Set Cover Hypothesis [Abb+22b; Cyg+16]. A natural question is whether there is an analog to Bringmann's Subset Sum algorithm for Knapsack. Namely, is there an algorithm for Knapsack in time $\widetilde{O}(n+W)$?

This question was answered negatively independently by Cygan, Mucha, Węgrzycki and Włodarczyk [Cyg+19] and Künnemann, Paturi and Schneider [KPS17]. They gave a conditional lower bound showing that there is no algorithm that solves Knapsack instances on n items with capacity $W = \Theta(n)$ in time $O(n^{2-\delta})$ for any $\delta > 0$. (We give more details about this result and its associated hardness hypothesis in Section 1.2.) This means that we cannot hope to match the running time of Bringmann's Subset Sum algorithm. Moreover, this shows that Bellman's dynamic programming algorithm in time $O(nW)$ is conditionally optimal.

However, this is not the end of the story. There are different ways in which one could circumvent this conditional lower bound to obtain faster algorithms than Bellman's in some parameter regimes. For example, it is open whether there is an $\widetilde{O}(n^2+W)$-time algorithm for Knapsack (or even in time $\widetilde{O}(n^2+W^{1.99})$). Observe that this running time is faster than Bellman's when $n \ll W$, and that the lower bound does not rule it out. Another way to overcome the lower bound is to measure the running time in terms of other parameters. Two of the most natural parameters are the maximum profit among the items $p_{\max}$, and the maximum weight $w_{\max}$. Note that we can assume without loss of generality that $p_{\max} \leq \mathrm{OPT}$ and $w_{\max} \leq W$. Therefore, a small polynomial dependence on these parameters can lead to faster algorithms compared to the standard dynamic programming algorithm on instances where $p_{\max} \ll \mathrm{OPT}$ or $w_{\max} \ll W$.

All of our results are driven by trying to develop faster Knapsack algorithms than the standard dynamic programming approach and/or circumventing the existing conditional lower bound.

1.1 Our Results

Our main contribution is a collection of algorithms for various Knapsack problems.

1.1.1 Pseudopolynomial Algorithms for Knapsack

Our first set of results are exact algorithms to solve Knapsack, where we parameterize the running time by the number of items n, the maximum weight $w_{\max}$, the knapsack capacity W, the maximum profit $p_{\max}$ and the optimal value of the instance OPT. Note that since any feasible solution includes at most all n items, we can assume without loss of generality that $W \leq n \cdot w_{\max}$ and $\mathrm{OPT} \leq n \cdot p_{\max}$. There is a long line of works that has developed algorithms for Knapsack using these parameters, starting with the seminal dynamic programming algorithm of Bellman which runs in time $O(n \min\{\mathrm{OPT}, W\})$. Table 1.1 contains a table of the known results.

Table 1.1: Pseudopolynomial-time algorithms for Knapsack. The results are displayed in chronological order.

Reference	Running Time
Bellman [Bel57]	$O(n \cdot \min\{W, \mathrm{OPT}\})$
Pisinger [Pis99]	$O(n \cdot p_{\max} \cdot w_{\max})$
Kellerer and Pferschy [KP04], also [Bat+18; AT19]	$\widetilde{O}(n + w_{\max} \cdot W)$
Bateni, Hajiaghayi, Seddighin and Stein [Bat+18]	$\widetilde{O}(n + p_{\max} \cdot W)$
Bateni, Hajiaghayi, Seddighin and Stein [Bat+18]	$\widetilde{O}((n + W) \cdot \min\{w_{\max}, p_{\max}\})$
Axiotis and Tzamos [AT19]	$\widetilde{O}(n \cdot \min\{w_{\max}^2, p_{\max}^2\})$
Polak, Rohwedder and Węgrzycki [PRW21]	$O(n + \min\{w_{\max}^3, p_{\max}^3\})$
Main Theorem 1.1	$\widetilde{O}(n + (W + \mathrm{OPT})^{1.5})$
Main Theorem 1.2	$\widetilde{O}(n \cdot w_{\max} \cdot p_{\max}^{2/3})$
Main Theorem 1.3	$\widetilde{O}(n \cdot p_{\max} \cdot w_{\max}^{2/3})$
He and Xu [HX24]	$\widetilde{O}(n^{1.5} \cdot \min\{p_{\max}, w_{\max}\})$
Jin [Jin23b], He and Xu [HX24]	$\widetilde{O}(n + \min\{w_{\max}, p_{\max}\}^{2.5})$
Chen, Lian, Mao and Zhang [Che+24]	$\widetilde{O}(n + \min\{w_{\max}, p_{\max}\}^{2.4})$
Bringmann [Bri23], Jin [Jin23a]	$\widetilde{O}(n + \min\{w_{\max}, p_{\max}\}^{2})$

Due to the interplay of all five parameters, some of the running times in Table 1.1 are incomparable. However, note that when $p_{\max} \approx w_{\max} \approx W \approx \mathrm{OPT} \approx n$, all existing algorithms up to 2021 require at least quadratic time $\Omega(n^2)$. In our first result, we give the first algorithm to break this quadratic barrier by considering the combined weight and profit parameter $W + \mathrm{OPT}$.

Main Theorem 1.1. *There is a randomized algorithm for Knapsack that runs in time*

$$\widetilde{O}(n + (W + \mathrm{OPT})^{1.5})$$

and succeeds with high probability.

We prove Main Theorem 1.1 in Section 3.3.

Next, we focus on the parameterization by the maximum profit $p_{\max}$ and the maximum weight $w_{\max}$ in the item set. Note that when $p_{\max} \approx w_{\max} \approx n$ and $W \approx \mathrm{OPT} \approx n^2$, all known algorithms up until 2022 require time $\Omega(n^3)$. In particular, up to this point, in this regime the algorithm in time $O(n w_{\max} p_{\max})$ of Pisinger from 1999 [Pis99] was the best known. Our next two results are the first to break this cubic barrier.

Main Theorem 1.2. *There is a randomized algorithm for Knapsack that runs in time*

$$\widetilde{O}((p_{\max}W)^{2/3}(n w_{\max})^{1/3} + n w_{\max})$$

and succeeds with high probability. Using the bound $W \le n w_{\max}$*, this running time is at most* $\widetilde{O}(n w_{\max} p_{\max}^{2/3})$*.*

Symmetrically, we obtain the following:

Main Theorem 1.3. *There is a randomized algorithm for Knapsack that runs in time*

$$\widetilde{O}((w_{\max}\mathrm{OPT})^{2/3}(n p_{\max})^{1/3} + n p_{\max})$$

and succeeds with high probability. Using the bound $\mathrm{OPT} \le n p_{\max}$*, this running time is at most* $\widetilde{O}(n p_{\max} w_{\max}^{2/3})$*.*

We prove Main Theorems 1.2 and 1.3 in Section 2.3

Follow-up Work As can be seen in Table 1.1, there has been a tremendous amount of progress in Knapsack *after* the publication of our works [BC22; BC23]. The chronology is as follows. On 17.05.2022 we published a preprint of [BC22] which contains Main Theorem 1.1. Later, on 02.05.2023, we published a preprint of [BC23] which proves Main Theorems 1.2 and 1.3. On 18.06.2023 Jin published [Jin23b], giving an algorithm in time $\widetilde{O}(n + \min\{w_{\max}, p_{\max}\}^{2.5})$. On 24.07.2023 Chen, Lian, Mao and Zhang published [Che+24], giving an algorithm in time $\widetilde{O}(n + \min\{w_{\max}, p_{\max}\}^{2.4})$, improving upon Jin's result. On 06.08.2023 Bringmann published [Bri23], giving an algorithm in time $\widetilde{O}(n + \min\{w_{\max}, p_{\max}\}^{2})$. On the next day, Jin published [Jin23a] independently obtaining the same running time as Bringmann (up to log factors). Bringmann's and Jin's results subsume the earlier results by Jin and Chen et al.'s. Finally, on 22.08.2023 He and Xu published a preprint of [HX24], giving two algorithms that run time $\widetilde{O}(n^{1.5}\min\{w_{\max}, p_{\max}\})$ and $\widetilde{O}(n + \min\{p_{\max}, w_{\max}\}^{2.5})$, respectively.

Combining He and Xu's [HX24] algorithm in time $\widetilde{O}(n^{1.5}\min\{w_{\max}, p_{\max}\})$ and Bringmann's [Bri23] and Jin's [Jin23a] algorithm in time $\widetilde{O}(n+\min\{p_{\max}, w_{\max}\}^{2})$ yields a strict improvement in all parameter regimes over our algorithms in time $\widetilde{O}(nw_{\max}p_{\max}^{2/3})$ and $O(np_{\max}w_{\max}^{2/3})$ given by Main Theorems 1.2 and 1.3.

However, our more complicated running time $\widetilde{O}((p_{\max}W)^{2/3}(n w_{\max})^{1/3} + n w_{\max})$ given by Main Theorem 1.2 (or its symmetric counterpart in Main Theorem 1.3) is still the best-known in some regimes. For example, when $w_{\max} \approx p_{\max}$ and $W = \Theta(n^{1.5})$, our running time becomes $\widetilde{O}(n^{1+1/3} w_{\max})$, which is faster than Bringmann's and Jin's result whenever $n^{1+1/3} \ll w_{\max}$.

Our algorithm in time $\widetilde{O}(n + (W + \mathrm{OPT})^{1.5})$ (Main Theorem 1.1) is generally incomparable to these later results. For example, as advertised earlier, in the regime when $w_{\max}, W, p_{\max}, \mathrm{OPT} = \Theta(n)$, our algorithm is still the only known result which runs in subquadratic time.

Finally, we remark that our techniques are generally different from these follow-up works.

1.1.2 Pseudopolynomial Algorithms for Unbounded Knapsack

For our next set of results, we study the closely related *Unbounded Knapsack* problem. Here, the input is the same as for Knapsack, namely a set $\mathcal{I}$ of n items with weights $w_1, \dots, w_n \in \mathbb{N}$ and profits $p_1, \dots, p_n \in \mathbb{N}$, along with a knapsack capacity $W \in \mathbb{N}$. The goal is to compute

$$\mathrm{OPT} := \max \left\{ \sum_{i=1}^{n} p_i x_i \mid \sum_{i=1}^{n} w_i x_i \leq W, x \in \mathbb{N}^n \right\}.$$

That is, the difference[2] is that now any item can be chosen an arbitrary number of times in a solution, i.e. $x \in \mathbb{N}^n$ instead of $x \in \{0, 1\}^n$.

Bellman's dynamic program extends to the unbounded setting seamlessly, and gives an algorithm in time $O(n \min\{W, \mathrm{OPT}\})$. Here, we shall be specifically concerned with algorithms parameterized by $p_{\max}$ and $w_{\max}$. Note that unlike Knapsack, it is not necessarily the case that $W \leq n \cdot w_{\max}$ and $\mathrm{OPT} \leq n \cdot p_{\max}$, since a solution might include more than n copies of an item. Therefore, obtaining an algorithm with only a polynomial dependence in n, $p_{\max}$ and $w_{\max}$ seems more challenging (in particular, it does not follow directly from the dynamic programming algorithm). Indeed, the first result of this type was by given by Tamir in 2009, who gave an algorithm for the problem in time $O(n^2 \min\{p_{\max}, w_{\max}\}^2)$ [Tam09]. A series of more recent works have progressively improved upon Tamir's algorithm, see Table 1.2 for an overview. To compare these running times, note that in Unbounded Knapsack we can assume that $n \leq w_{\max}$ without loss of generality since if there are multiple items with the same weight, we can discard all except the one with the largest profit. Similarly, we can assume without loss of generality that $n \leq p_{\max}$. With this in mind, observe that the best known running time previous to our work is due to Chan and He [CH22] who gave an algorithm in time $\widetilde{O}(n \min\{w_{\max}, p_{\max}\})$.

Note that when $w_{\max} \approx p_{\max} \approx n$ all algorithms in Table 1.2 require at least quadratic time $\Omega(n^2)$. In our next result, we overcome this quadratic barrier by considering the combined parameter $w_{\max} + p_{\max}$.

Main Theorem 1.4. *There is a randomized algorithm for Unbounded Knapsack that runs in expected time* $\widetilde{O}(n + (p_{\max} + w_{\max})^{1.5})$.

We prove Main Theorem 1.4 in Section 3.2.

1.1.3 Approximation Schemes for Unbounded Knapsack

Since Unbounded Knapsack is well known to be NP-hard [Kar72], it is natural to study approximation algorithms. In particular, a *fully polynomial-time approximation scheme* (FPTAS) given $0 < \varepsilon < 1$ computes a solution with total weight at most W (i.e. a feasible

2. Some works refer to Knapsack as "0-1 Knapsack" to distinguish it from Unbounded Knapsack.

Table 1.2: Pseudopolynomial-time algorithms for Unbounded Knapsack. The results are displayed in chronological order.

Reference	Running Time
Bellman [Bel57]	$O(n \cdot \min\{W, \mathrm{OPT}\})$
Tamir [Tam09]	$O(n^2 \cdot w_{\max}^2)$
Bateni, Hajiaghayi, Seddighin and Stein [Bat+18]	$\widetilde{O}(n \cdot w_{\max}^2)$
Eisenbrand and Weismantel [EW20]	$O(n \cdot \min\{w_{\max}^2, p_{\max}^2\})$
Axiotis and Tzamos [AT19]	$\widetilde{O}(n + \min\{w_{\max}^2, p_{\max}^2\})$
Jansen and Rohwedder [JR23]	$\widetilde{O}(n + \min\{w_{\max}^2, p_{\max}^2\})$
Chan and He [CH22]	$\widetilde{O}(n \cdot \min\{w_{\max}, p_{\max}\})$
Main Theorem 1.4	$\widetilde{O}(n + (w_{\max} + p_{\max})^{1.5})$

solution) and total profit at least $(1-\varepsilon)\mathrm{OPT}$ in time $\mathrm{poly}(n, 1/\varepsilon)$. The first FPTAS for Unbounded Knapsack was designed by Ibarra and Kim in 1975 [IK75] and runs in time $\widetilde{O}(n + (1/\varepsilon)^4)$. In 1979 Lawler [Law79] improved the running time to $\widetilde{O}(n + (1/\varepsilon)^3)$. This was the best known until Jansen and Kraft in 2018 [JK18] presented an FPTAS running in time $\widetilde{O}(n + (1/\varepsilon)^2)$. This algorithm has a matching conditional lower ruling out time $O((n + 1/\varepsilon)^{2-\delta})$ for any $\delta > 0$ [Cyg+19; KPS17; MWW19].

Our next result is a new FPTAS for Unbounded Knapsack which (we believe) is simpler than Jansen and Kraft's, and has a lower order improvement in the running time:

Theorem 1.5. *Unbounded Knapsack has a deterministic FPTAS that runs in time*

$$\widetilde{O}\left(n + \frac{(1/\varepsilon)^2}{2^{\Omega(\sqrt{\log(1/\varepsilon)})}}\right).$$

Weak Approximation for Unbounded Knapsack Motivated by the matching upper and conditional lower bounds for FPTASs for Unbounded Knapsack, we study the relaxed notion of *weak approximation* as coined in [MWW19]: we relax the weight constraint and seek a solution with total weight at most $(1+\varepsilon)W$ and total profit at least $(1-\varepsilon)\mathrm{OPT}$. Note that OPT is still the optimal value of any solution with weight at most W. This can be interpreted as bicriteria approximation (approximating both the weight and profit constraint) or as resource augmentation (the optimal algorithm is allowed weight W while our algorithm is allowed a slightly larger weight of $(1+\varepsilon)W$). All of these are well-established relaxations of the standard (=strong[3]) notion of approximation. Such weaker notions of approximation are typically studied when a PTAS for the strong notion of approximation is not known. More generally, studying these weaker notions

3. By "strong" approximation we mean the standard (non-weak) notion of approximation.

is justified whenever there are certain limits for strong approximations, to see whether these limits can be overcome by relaxing the notion of approximation. In particular, we want to understand whether this relaxation can overcome the conditional lower bound ruling out time $O((n+1/\varepsilon)^{2-\delta})$ for any $\delta > 0$. For the related Subset Sum problem this question has been resolved positively: Bringmann and Nakos [BN21] conditionally ruled out strong approximation algorithms in time $O((n+1/\varepsilon)^{2-\delta})$ for any $\delta > 0$, but Mucha, Węgrzycki and Włodarczyk [MWW19] designed a weak FPTAS in time $\widetilde{O}(n+(1/\varepsilon)^{5/3})$, which was subsequently improved to time $\widetilde{O}(n+(1/\varepsilon)^{3/2})$ by Wu and Chen [WC22].

Our next result gives a positive answer for Unbounded Knapsack:

Main Theorem 1.6. *Unbounded Knapsack has a weak approximation scheme running in expected time* $\widetilde{O}(n+(\frac{1}{\varepsilon})^{1.5})$.

We prove Theorem 1.5 and Main Theorem 1.6 in Section 3.4.

Related Work: FPTAS for Knapsack Very recently (specifically, *after* the publication of our papers [BC22; BC23]) Mao [Mao23] and Chen, Lian, Mao and Zhang [Che+23] independently gave FPTASs for Knapsack in time $\widetilde{O}(n+1/\varepsilon^2)$, which is conditionally optimal. This culminated a line of works that aimed to obtain quadratic dependence on $1/\varepsilon$ [IK75; Law79; KP04; Rhe15; Cha18; Jin19; DJM23].

1.2 Min-Plus Convolution

All of our results stated in Section 1.1 are obtained by studying Knapsack through its intimate connection to *min-plus convolution*. Given functions $f, g\colon [n] \mapsto \mathbb{Z}$, their min-plus convolution is the function $h\colon [2n] \mapsto \mathbb{Z}$ defined as $h(x) = \min_{x'} f(x') + g(x-x')$ for $x \in [2n]$. Simply evaluating this definition, yields an algorithm to compute the min-plus convolution in time $O(n^2)$. This running time can be improved to $n^2/2^{\Omega(\sqrt{\log n})}$-time via a reduction to min-plus matrix product due to Bremner et al. [Bre+14], and using Williams' algorithm for the latter [Wil18] (which was derandomized later by Chan and Williams [CW21]). The lack of faster algorithms has led to the *Min-Plus Convolution Hypothesis*, which postulates that there is no algorithm in time $O(n^{2-\delta})$ for any $\delta > 0$ for this problem [Cyg+19; KPS17]. Many problems have lower bounds conditioned on this hypothesis, see for example [BIS17; CH21; Cyg+19; JR23; KPS17; LRC14; BN21; Kle22; KP23].

Central to our work is a reduction from min-plus convolution to Unbounded Knapsack shown independently by Cygan, Mucha, Węgrzycki and Włodarczyk [Cyg+19] and Künnemann, Paturi and Schneider [KPS17]. In particular, they showed that if Unbounded Knapsack with n items and knapsack capacity $W = O(n)$ can be solved in subquadratic time, then min-plus convolution can be solved in subquadratic time. This reduction immediately implies matching conditional lower bounds for some of the known exact algorithms for Unbounded Knapsack with running times $O(nW)$ [Bel57], $\widetilde{O}(w_{\max}^2)$ [JR23; AT19] and $\widetilde{O}(n\,w_{\max})$ [CH22].

The same reduction extends to the dual case, i.e., an exact subquadratic-time algorithm for Unbounded Knapsack with $\mathrm{OPT} = O(n)$ would result in a subquadratic-time algorithm for min-plus convolution. This establishes matching conditional lower bounds for the algorithms in time $O(n \cdot \mathrm{OPT})$ [Bel57], $\widetilde{O}(p_{\max}^2)$ [JR23; AT19] and $\widetilde{O}(n\, p_{\max})$ [CH22]. Moreover, by setting $\varepsilon = \Theta(1/\mathrm{OPT})$, an FPTAS for Unbounded Knapsack would yield an exact algorithm for min-plus convolution, establishing that the $\widetilde{O}(n + (1/\varepsilon)^2)$-time FPTAS by Jansen and Kraft [JK18] is conditionally optimal. This last observation was pointed out in [MWW19].

The same reduction is also known for Knapsack [Cyg+19; KPS17], so a similar discussion applies to Knapsack. In particular, the algorithms due to Bringmann [Bri23] and Jin [Jin23a] in time $\widetilde{O}(n + \min\{w_{\max}, p_{\max}\}^2)$ are conditionally optimal.

Circumventing the Lower Bound By inspecting the conditional lower bound of Cygan et al. [Cyg+19] and Künnemann et al. [KPS17], we observe that the reduction constructs hard instances of knapsack where only the profit parameters or only the weight parameters are under control; one of the two must be very large to obtain a hardness reduction. In more detail, the reduction produces instances with n items where $p_{\max}, \mathrm{OPT} = O(n)$ but $w_{\max}, W = \Omega(n^2)$, and (conditionally) rules out algorithms in time $O(n^{2-\delta})$ for any $\delta > 0$. Some of our results exploit this observation to circumvent the lower bound:

- To obtain Main Theorem 1.4, we consider the combined profit and weight parameter $w_{\max} + p_{\max}$. Since the reduction produces instances where $w_{\max} = \Omega(n^2)$, a subquadratic algorithm in terms of $w_{\max} + p_{\max}$ does not contradict the quadratic lower bound.

- Similarly, for Main Theorem 1.1 we consider the combined parameter $\mathrm{OPT} + W$. Our result does not contradict the lower bound since the reduction produces instances with weight budget $W = \Omega(n^2)$.

- For our weak approximation scheme given by Main Theorem 1.6, the usage of resource augmentation allows us to bypass the lower bound. More precisely, recall that the weak approximation scheme is allowed to find a solution with total weight at most $(1 + \varepsilon)W$. Thus, to obtain a feasible solution (i.e. a solution with weight at most W) by running such weak approximation scheme as a black-box, we have to set $\varepsilon = \Theta(1/W)$. Since the reduction produces instances with $W = \Omega(n^2)$, this means that a subquadratic algorithm in terms of $1/\varepsilon$ does not contradict the lower bound.

1.2.1 Faster algorithms via Structured Min-Plus Convolution

Despite the Min-Plus Convolution Hypothesis, there are structured instances of min-plus convolution that can be solved in subquadratic time [Agg+87; Bat+18; Bus+94; CL15; Chi+22]. Some of these improvements have been instrumental to obtain many of the Knapsack and Unbounded Knapsack algorithms listed in Table 1.1 and Table 1.2:

- When one of the functions is convex, their min-plus convolution can be computed in time $O(n)$ using the SMAWK algorithm [Agg+87]. This has been used for Knapsack indirectly[4] by Kellerer and Pferschy [KP04], and explicitly by Axiotis and Tzamos [AT19], Polak, Rohwedder and Węgrzycki [PRW21], Chen, Lian, Mao and Zhang [Che+24], Jin [Jin23b; Jin23a] and Bringmann [Bri23].

- Bateni, Hajiaghayi, Seddighin and Stein [Bat+18] introduced the *prediction technique* to show that the min-plus convolution of certain instances arising from Knapsack can be computed efficiently. More precisely, let h be the min-plus convolution of two given functions $f, g\colon [n] \mapsto \mathbb{Z}$. They show that if one is given n intervals $[x_i \,..\, y_i]$ for $i \in [n]$ satisfying (i) $|h(i+j) - (f(i) + g(j))| \leq \Delta$ for every $i \in [n]$ and $j \in [x_i \,..\, y_i]$, (ii) for every output $h(k)$ there exists at least one i such that $f(i) + g(k-i) = h(k)$ and $k - i \in [x_i \,..\, y_i]$ and (iii) $0 \leq x_i, y_i < n$ for all intervals and $x_i \leq x_j, y_i \leq y_j$ for all $i < j$; then h can be computed in time $\widetilde{O}(n \cdot \Delta)$. They showed that this is applicable in the context of Knapsack and Unbounded Knapsack and used their technique to obtain various algorithms (see Table 1.1 and Table 1.2).

All of our results for Knapsack and Unbounded Knapsack fall into the same category of improvements. More precisely, we (i) design a new algorithm for instances that are near-convex that naturally arise in the context of Knapsack, and (ii) show how to apply a known (subquadratic) algorithm for bounded and monotone instances of min-plus convolution for Knapsack. We expand on these two improvements below.

Near Convex MinPlus Convolution To prove Main Theorem 1.2 and Main Theorem 1.3 we design an efficient algorithm for a new class of structured instances of min-plus convolution, namely *near convex* functions: We say that $f\colon [n] \mapsto \mathbb{Z}$ is Δ-near convex, if there is a convex function $\breve{f}\colon [n] \mapsto \mathbb{Q}$ such that $\breve{f}(i) \leq f(i) \leq \breve{f}(i) + \Delta$ for all $i \in [n]$. Our theorem reads as follows:

Main Theorem 1.7 (Near Convex MinPlus Convolution)**.** *Let $f\colon [n] \mapsto [-U \,..\, U]$, and $g\colon [m] \mapsto [-U \,..\, U]$ be given as inputs where $n, m, U \in \mathbb{N}$. Let $\Delta \geq 1$ such that both f and g are Δ-near convex. Then the min-plus convolution of f and g can be computed in deterministic time $\widetilde{O}((n+m) \cdot \Delta)$.*

We view this as a replacement for the prediction technique by Bateni et al. [Bat+18]. Indeed, all uses of the prediction technique exploit near-convexity to ensure its preconditions, and thus all uses that we are aware of can be replaced by our Main Theorem 1.7. Since the prediction technique is both difficult to state and difficult to apply, we view Main Theorem 1.7 as replacing the prediction technique by an *easily applicable tool with a concise statement.* To showcase its usefulness, we apply this new tool to prove Main Theorems 1.2 and 1.3, making progress on Knapsack. We expect it to have wider applicability. We prove Main Theorem 1.7 in Section 2.2.

4. Kellerer and Pferschy did not use SMAWK, but gave a different algorithm for computing the min-plus convolution of these instances in time $O(n \log n)$.

Bounded Monotone Min-Plus Convolution A central special case of min-plus convolution that we study is when the input functions $f, g \colon [n] \mapsto [O(n)]$ are monotone non-decreasing, and have bounded domain. We call the task of computing the min-plus convolution of such sequences Bounded Monotone MinPlus Conv.

In a breakthrough result, Chan and Lewenstein gave an algorithm for Bounded Monotone MinPlus Conv that runs in expected time $O(n^{1.859})$ [CL15]. As a big hammer, their algorithm uses the famous Balog-Szemerédi-Gowers theorem from additive combinatorics in a beautiful algorithmic way. Recently, Chi, Duan, Xie and Zhang showed how to avoid this big hammer and improved the running time to expected $\widetilde{O}(n^{1.5})$ [Chi+22] via a simple and elegant algorithm.

To prove Main Theorem 1.1, Main Theorem 1.4 and Main Theorem 1.6 we use the algorithm of Chi, Duan, Xie and Zhang for Bounded Monotone MinPlus Conv as a subroutine. More precisely, our algorithms are phrased as *reductions* from various knapsack problems to Bounded Monotone MinPlus Conv. Therefore, any future improvements on Bounded Monotone MinPlus Conv immediately carries over to our results (replacing the exponent 1.5 by whatever is the improvement). Moreover, we complement our results with reductions in the opposite direction–thereby establishing some form of equivalence between Bounded Monotone MinPlus Conv and various parameterizations of Knapsack, see Section 3.5 for details.

1.3 Open Problems

We leave some open problems about Knapsack and related problems that we find most exciting:

1. *Subset Sum parameterized by the smallest number.*
   In Subset Sum we are given a set of numbers X and a target number t, the task is to decide if a subset of X sums to t. For the unbounded case, where the goal is to find whether a *multiset* of items in X sums to t, Jansen and Rohwedder [JR23] gave an algorithm in time $\widetilde{O}(n+u)$ where u is the largest number in the input. For the more standard "0-1" case where we ask for a subset of X summing to t, the best known running times are $\widetilde{O}(n+t)$ [Bri17; JW19], $O(nu)$ [Pis99], $\widetilde{O}(n+u^2/n)$ by combining [GM91] and [Bri17; JW19], and $\widetilde{O}(n+u^{3/2})$ by combining [GM91] and [Bri17; JW19] and [Pis99]; see also [BW21; PRW21] for generalizations to X being a multiset and related results. Is there an algorithm in time $\widetilde{O}(n+u)$?

2. *Even faster Knapsack algorithms.*
   The recent works of Bringmann [Bri23] and Jin [Jin23a] gave conditionally optimal algorithms for Knapsack in time $\widetilde{O}(n+\min\{w_{\max}, p_{\max}\}^2)$. Can they be improved to time $\widetilde{O}(n\min\{w_{\max}, p_{\max}\})$? This would match the best known algorithm for Unbounded Knapsack due to Chan and He [CH22].

3. *Subquadratic Knapsack algorithm.*
   Is there an algorithm for Knapsack in time $\widetilde{O}(n+(w_{\max}+p_{\max})^{1.999})$, in the spirit

of our result for Unbounded Knapsack (Main Theorem 1.4)? This would break the current quadratic barrier in the regime when $w_{\max}, p_{\max} = \Theta(n)$.

4. *Weak approximation for Knapsack.*
   We showed that it is possible to get a subquadratic (in terms of $1/\varepsilon$) weak FPTAS for Unbounded Knapsack (Main Theorem 1.6). Can this be extended to Knapsack? That is, is there a weak FPTAS for Knapsack in time $\widetilde{O}(n + 1/\varepsilon^{1.999})$?

5. *Improved near-convex min-plus convolution.*
   Our min-plus convolution for near convex functions (Main Theorem 1.7) assumes that both functions are near-convex. Can it be generalized to assume that only one function is near-convex? This would constitute a natural generalization of the SMAWK algorithm which computes the min-plus convolution of one convex function and an arbitrary one in linear time [Agg+87].

6. *Derandomization.*
   The algorithms given by Main Theorems 1.1, 1.2, 1.4 and 1.6 are all randomized. If we insist on deterministic algorithms, we note that by applying Chan and Lewenstein's deterministic $\widetilde{O}(n^{1.864})$-time algorithm for Bounded Monotone MinPlus Conv [CL15], we can obtain deterministic versions of Main Theorem 1.4 and Main Theorem 1.6 with exponent 1.864 instead of 1.5 (i.e. the only part where we use randomness is in applying Chi, Duan, Xie and Zhang's algorithm [Chi+22]). On the other hand, we do not know how to derandomize Main Theorem 1.1 and Main Theorem 1.2. As a concrete barrier, we note that Main Theorem 1.1 algorithm closely follows Bringmann's algorithm for Subset Sum [Bri17], whose derandomization is a notorious open problem.

7. *Improved algorithms for ILPs.*
   Consider an ILP of the form
   $$\max\{\, c^T x \mid Ax = b, x \in \mathbb{Z}^n_{\geq 0} \,\},$$
   where $A \in \mathbb{Z}^{m \times n}, c \in \mathbb{Z}^n$ and $b \in \mathbb{Z}^n$. Let Δ be an upper bound on the magnitude of the largest entry in A. Here, we consider m to be a fixed constant, and we are interested in algorithms parameterized by Δ and n. When $m = 1$, this ILP corresponds to Unbounded Knapsack. Jansen and Rohwedder gave an algorithm to solve these ILPs in time $\widetilde{O}(n + \Delta^{2m})$, and proved a matching conditional lower bound [JR23].

   However, consider ILPs where each entry of x has an upper bound, i.e. ILPs of the form
   $$\max\{\, c^T x \mid Ax = b, x \in \mathbb{Z}^n_{\geq 0}, x \leq u \,\}, \tag{1.1}$$
   where $A \in \mathbb{Z}^{m \times n}, c \in \mathbb{Z}^n$, $u \in \mathbb{Z}^n_{\geq 0}$ and $b \in \mathbb{Z}^n$. In this case, the best algorithm is due to Eisenbrand and Weismantel and runs in time $\widetilde{O}(n\Delta^{m(m+1)})$ [EW20]. Is there an algorithm for ILPs of the form (1.1) in time $\widetilde{O}(n + \Delta^{2m})$?[5] For the special case

5. Obtaining time $\widetilde{O}(n + \Delta^{O(m)})$ instead of $\widetilde{O}(n + \Delta^{O(m^2)})$ would already be of interest.

of Knapsack (i.e. when $m = 1$ and $u = \mathbb{1}^n, c \in \mathbb{Z}^n_{\geq 0}$), this question was answered affirmatively by Bringmann [Bri23] and Jin's [Jin23a] algorithms in time $\widetilde{O}(n + \Delta^2)$[6]. It is a tantalizing open problem to understand whether their techniques extend to general ILPs of the form (1.1).

1.4 Organization

The rest of this part of the thesis is organized as follows. In Section 1.5 we fix some notation that will be used throughout. Then, we split our results in two chapters: In Chapter 2 we give our new algorithm for near-convex min-plus convolution, and give an algorithm for Knapsack based on it. In Chapter 3 we give our results that are obtained via the usage of bounded monotone min-plus convolution.

1.5 Notation

In what follows, we fix some notation and state some preliminaries that will be used throughout this part of the thesis.

We denote the integers by $\mathbb{Z}$ and the nonnegative integers by $\mathbb{N}$. For $t \in \mathbb{N}$ we let $[t] = \{ 0, 1, \dots, t \}$. We use the notation $\operatorname{poly}(n) = n^{O(1)}$ and $\operatorname{polylog}(n) = (\log n)^{O(1)}$.

Let $A \in \mathbb{Z}^{n+1}$ be an integer sequence, i.e., $A[i] \in \mathbb{Z}$ for $i \in [n]$. Sometimes we will refer to such a sequence as a *function* $A\colon [n] \mapsto \mathbb{Z}$. With this in mind, we use the notation $-A$ to denote the entry-wise negation of A.

Given $a, b \in \mathbb{R}$ with $a \leq b$, we define

$$[a \mathinner{..} b] := \{ \max(0, \lfloor a \rfloor), \max(0, \lfloor a \rfloor) + 1, \dots, \lceil b \rceil - 1, \lceil b \rceil \}.$$

The non-standard rounding and capping at 0 in the definition of $[a \mathinner{..} b]$ is useful to index a subsequence $A[a \mathinner{..} b]$ when a and b might not be nonnegative integers.

We formally recap the definitions of the main problems we study, Knapsack and Unbounded Knapsack.

Problem 1.8 (Knapsack). *Given a set of n items $\mathcal{I}$ with weights $w_1, \dots, w_n \in \mathbb{N}$ and profits $p_1, \dots, p_n \in \mathbb{N}$ along with a knapsack capacity $W \in \mathbb{N}$. The task is to compute* $\max\{ \sum_i p_i x_i \mid \sum_i w_i x_i \leq W, x \in \{0, 1\}^n \}$.

Problem 1.9 (Unbounded Knapsack). *Given a set of n items $\mathcal{I}$ with weights $w_1, \dots, w_n \in \mathbb{N}$ and profits $p_1, \dots, p_n \in \mathbb{N}$ along with a knapsack capacity $W \in \mathbb{N}$. The task is to compute* $\max\{ \sum_i p_i x_i \mid \sum_i w_i x_i \leq W, x \in \mathbb{N}^n \}$.

6. In fact, Bringmann's algorithm works for the more general case of $u \in \mathbb{Z}^n_{\geq 0}$ [Bri23], which is called "Bounded Knapsack".

Max-plus convolution. The max-plus convolution of two sequences $A[0\,..\,n] \in \mathbb{Z}^{n+1}, B[0\,..\,m] \in \mathbb{Z}^{m+1}$, denoted by $\textsc{MaxConv}(A, B)$, is a sequence of length $n + m + 1$ where for each $k \in [n + m]$ we have $\textsc{MaxConv}(A, B)[k] := \max_{i+j=k} A[i] + B[j]$ (we interpret out-of-bounds entries as $-\infty$). The min-plus convolution $\textsc{MinConv}(A, B)$ is defined analogously, but replacing max by a min. Note that by negating the entries of the sequences, these two operations are equivalent.

Fact 1.10. *For any $A \in \mathbb{Z}^{n+1}, B \in \mathbb{Z}^{m+1}$, we have $\textsc{MaxConv}(A, B) = -\textsc{MinConv}(-A, -B)$.*

We will use the following handy notation: Given sequences $A[0\,..\,n], B[0\,..\,n]$ and intervals $I, J \subseteq [n]$ and $K \subseteq [2n]$, we denote by $C[K] := \textsc{MaxConv}(A[I], B[J])$ the computation of $C[k] := \max\{\, A[i] + B[j] \mid i \in I, j \in J, i + j = k \,\}$ for each $k \in K$.

Proposition 1.11. *If max-plus convolution of length-n sequences can be solved in time $T(n)$, then $C[K] = \textsc{MaxConv}(A[I], B[J])$ can be computed in time $O(T(|I| + |J|) + |K|)$.*

Proof. After shifting the indices, we can assume that $A[I]$ and $B[J]$ start from the index 0, i.e., $A'[0\,..\,|I| - 1]$ and $B'[0\,..\,|J| - 1]$. Compute $C' := \textsc{MaxConv}(A', B')$ in time $T(|A| + |B|)$. By shifting the indices back, we can infer the values of the entries $C[I + J] = \textsc{MaxConv}(A[I], B[J])$. Thus, we can simply read off the entries in $C[K]$ from the array C'. □

Machine Model We work with the standard word RAM model of computation. Given as input an item set $\mathcal{I}$ with weights $w_1, \dots, w_n \in \mathbb{N}$ profits $p_1, \dots, p_n \in \mathbb{N}$, and knapsack capacity $W \in \mathbb{N}$, we assume the words have size $\Theta(\log(n) + \log(W) + \log(p_{\max}))$. Here, $p_{\max}$ is the largest profit in the item set, i.e. $p_{\max} = \max_i p_i$. That is, we assume we can store the total weight and total profit of any feasible solution to the given knapsack instance in a single machine word.

2 Algorithms via Near Convex Min-Plus Convolution

This chapter contains our algorithm for min-plus convolution for near-convex functions and its application to Knapsack. The content is based on our publication [BC23].

Organization The outline of this chapter is as follows. We start with brief preliminaries in Section 2.1. In Section 2.2 we present our algorithm for min-plus convolution, proving Main Theorem 1.7. Finally, in Section 2.3 we apply this result to obtain our Knapsack algorithms, proving Main Theorem 1.2 and Main Theorem 1.3.

2.1 Preliminaries

We say that a function $f\colon [n] \mapsto \mathbb{Q}$ is *convex* if $f(i) - f(i-1) \leq f(i+1) - f(i)$ holds for every $i \in [1\,..\,n-1]$. We say that f is *concave* if $-f$ is convex.

Definition 2.1 (Near Convex and Near Concave Functions). *For $\Delta \geq 0$, we say that a function $f\colon [n] \mapsto \mathbb{Z}$ is Δ-near convex, if there is a convex function $\breve{f}\colon [n] \mapsto \mathbb{Q}$ such that $\breve{f}(i) \leq f(i) \leq \breve{f}(i) + \Delta$. We say that f is Δ-near concave if the function $-f$ is Δ-near convex.*

If the input consists of N numbers in $[-U\,..\,U]$, we denote $\widetilde{O}(T) = \bigcup_{c\geq 0} O(T \log^c(NU))$.

2.2 MinPlus Convolution for Near-Convex Sequences

In this section we prove Main Theorem 1.7, which we restate for convenience.

Main Theorem 1.7 (Near Convex MinPlus Convolution). *Let $f\colon [n] \mapsto [-U\,..\,U]$, and $g\colon [m] \mapsto [-U\,..\,U]$ be given as inputs where $n, m, U \in \mathbb{N}$. Let $\Delta \geq 1$ such that both f and g are Δ-near convex. Then the min-plus convolution of f and g can be computed in deterministic time $\widetilde{O}((n+m)\cdot\Delta)$.*

Before diving into the technical details, we start with a high level overview.

Proof Overview Let $f, g\colon [n] \mapsto \mathbb{Z}$ be the input functions, and let h be their min-plus convolution, which we aim to compute. First we observe that we can obtain the convex approximations $\breve{f}, \breve{g}$ witnessing the Δ-near convexity of f and g, and compute their min-plus convolution $\breve{h}$ efficiently (see Lemma 2.7). By exploiting $\breve{h}$ and the convexity of $\breve{f}$ and $\breve{g}$, we identify a structured set $R \subseteq [n]^2$ with the property that any $(i, j) \in [n]^2 \setminus R$ satisfies $f(i) + g(j) > h(i + j)$. Then, we give a simple recursive algorithm to cover R with a collection C of disjoint dyadic boxes $I \times J$, where $(I, J) \in C$ satisfies $I, J \subseteq [n]$ and $I \times J \subseteq R$. Thus, we can infer h by computing the *sumset*

$$A := \{\, (i, f(i)) \mid i \in I \,\} + \{\, (j, g(j)) \mid j \in J \,\}$$

and taking $h(k) = \min\{\, y \mid (k, y) \in A \,\}$ for every $(I, J) \in C$ (see Algorithm 2). To implement this plan efficiently, we observe that inside I and J, the functions $f[I]$ and $g[J]$ are close to linear functions with the same slope up to an additive error of $\pm O(\Delta)$ (which follows from their Δ-near convexity, see Lemma 2.9). This implies that their sumset is small; more precisely it has size $O((|I| + |J|)\Delta)$ (see Lemma 2.10). Finally, we make use of known tools that can compute a sumset in time proportional to its size (see Theorem 2.5).

The idea of identifying a covering with small sumsets to efficiently compute the min-plus convolution is inspired by Chan and Lewenstein's [CL15] algorithm for computing the min-plus convolution of bounded monotone sequences (in which they do not use convexity in any form). Our algorithm shares some similarities with the prediction technique by Bateni, Hajiaghayi, Seddighin and Stein [Bat+18]. In particular, the covering by dyadic boxes where functions are near-linear resembles the way in which they exploit the intervals $[x_i \mathinner{..} y_i]$ required by their algorithm.

2.2.1 Preparations

Throughout this section, fix the functions $f\colon [n] \mapsto [-U \mathinner{..} U], g\colon [m] \mapsto [-U \mathinner{..} U]$. Recall that we say that $f\colon [n] \mapsto \mathbb{Z}$ is Δ_f-near convex, if there is a convex function $\breve{f}\colon [n] \mapsto \mathbb{Q}$ such that $\breve{f}(i) \le f(i) \le \breve{f}(i) + \Delta_f$ for all $i \in [n]$ (see Definition 2.1). First observe that the lower convex hull of the points $\{\, (i, f(i)) \mid i \in [n] \,\}$ gives the pointwise maximal convex function $\breve{f}$ with $\breve{f} \le f$. This can be computed in time $O(n)$ by Graham's scan [Gra72], since the points are already sorted by x-coordinate. Then, we can infer $\Delta_f = \max\{\, 1, \max_{i \in [n]} f(i) - \breve{f}(i) \,\}$. Thus, from now on we assume that we know $\breve{f}, \Delta_f, \breve{g}, \Delta_g$. Set $\Delta := \max\{\, \Delta_f, \Delta_g \,\}$. Let $\breve{h} := \textsc{MinConv}(\breve{f}, \breve{g})$ and $h := \textsc{MinConv}(f, g)$. The goal is to compute h.

Let's start by introducing some notation. We call $(i, j) \in [n] \times [m]$ a *point*. We visualize a point (i, j) as lying on the i-th row and j-th column of an $n \times m$ grid, where $(0, 0)$ is on the bottom-left corner and (n, m) on the top right corner. A point (i, j) lies on *diagonal* $i + j$. For any $\delta \ge 0$, a point (i, j) is δ-*relevant* if $\breve{f}(i) + \breve{g}(j) \le \breve{h}(i + j) + \delta$. We denote by R_δ the set of all δ-relevant points.

Points that are 0-relevant are important because of the following observation: We call i a *witness* for $\breve{h}(k)$ if $\breve{f}(i) + \breve{g}(k - i) = \breve{h}(k)$. Thus, observe that i is a witness for $\breve{h}(k)$ if and only if $(i, k - i)$ is a 0-relevant point.

The importance of 2Δ-relevant points is captured by the following lemma:

Lemma 2.2. *If $(i, j) \notin R_{2\Delta}$ then $f(i) + g(j) > h(i + j)$.*

That is, points that are not 2Δ-relevant can be ignored for the purpose of computing h.

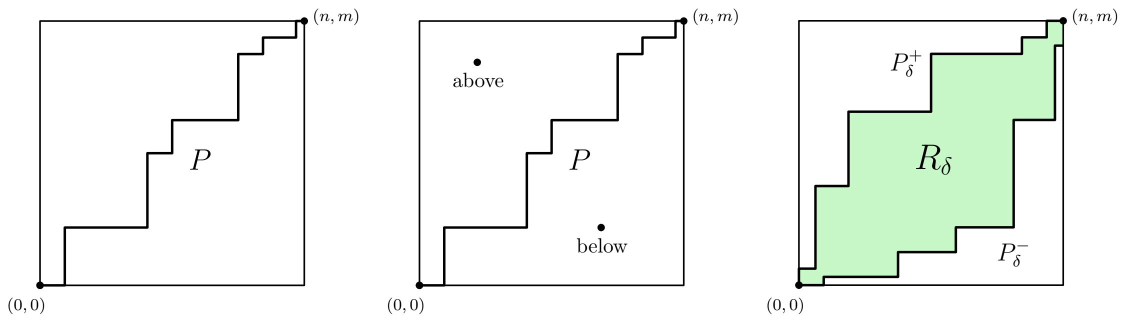
Proof. Since (i, j) is not 2Δ-relevant, it holds that $f(i)+g(j) \geq \breve{f}(i)+\breve{g}(j) > \breve{h}(i+j)+2\Delta$. Let $k := i + j$, and let i^* be a witness for $\breve{h}(k)$, i.e., $\breve{f}(i^*) + \breve{g}(k - i^*) = \breve{h}(k)$. Then,

$$h(k) \leq f(i^*) + g(k - i^*) \leq \breve{f}(i^*) + \Delta + \breve{g}(k - i^*) + \Delta = \breve{h}(k) + 2\Delta < f(i) + g(j). \quad \square$$

We say that a set of points P is a *monotone path* if for every $k \in [n + m]$ P contains exactly one point (i_k, j_k) on diagonal k, and we have $(i_{k+1}, j_{k+1}) \in \{ (i_k{+}1, j_k), (i_k, j_k{+}1) \}$ for every $k \in [n + m - 1]$, see Figure 2.1a for an illustration. For any $\delta > 0$, we let

$$P^+_\delta := \{ (i, k - i) \mid k \in [n + m], i \in [n] \text{ is maximal s.t. } (i, k - i) \text{ is } \delta\text{-relevant} \},$$
$$P^-_\delta := \{ (i, k - i) \mid k \in [n + m], i \in [n] \text{ is minimal s.t. } (i, k - i) \text{ is } \delta\text{-relevant} \}.$$

The next two lemmas show that P^+_δ, P^-_δ are monotone paths and that P^+_δ, P^-_δ form the boundary of R_δ, see Figure 2.1c for an illustration. This establishes the structure of R_δ that we will exploit later.



(a) A monotone path P (b) Points above and below P (c) R_δ is between P^+_δ and P^-_δ

Figure 2.1: Visualizations for concepts used in Section 2.2.

Lemma 2.3 (Monotone Paths). *For any $\delta \geq 0$, P^-_δ, P^+_δ are monotone paths.*

Proof. Since for each $k \in [n+m]$, $\breve{h}(k)$ has a witness, there is a 0-relevant point $(i, k - i)$. Since every 0-relevant point is also δ-relevant, it follows that P^-_δ contains exactly one point (i_k, j_k) with $i_k + j_k = k$ for every $k \in [n + m]$.

In the following, we show that $i_{k-1} \leq i_k \leq i_{k-1} + 1$ holds for all $k \in [1\,..\,n + m]$. Since $i_k + j_k = k$, it then also follows that $k - 1 - j_{k-1} \leq k - j_k \leq k - 1 - j_{k-1} + 1$, which yields $j_{k-1} \leq j_k \leq j_{k-1} + 1$. Since $i_{k+1} + j_{k+1} = k + 1 = i_k + j_k + 1$, it follows that $(i_k, j_k) \in \{ (i_{k-1} + 1, j_{k-1}), (i_{k-1}, j_{k-1} + 1) \}$. So it remains to prove $i_{k-1} \leq i_k \leq i_{k-1} + 1$. We distinguish two cases.

Case 1: $i_k \leq i_{k-1}$. We show that in this case $i_k \geq i_{k-1}$ (and thus $i_k = i_{k-1}$). Let i^*_{k-1} be a witness for $\breve{h}(k-1)$. Note that $i^*_{k-1} \geq i_{k-1}$ by definition of P^-_δ. We have

$$\breve{f}(i_k) + \breve{g}(k - i_k) \leq \breve{h}(k) + \delta \leq \breve{f}(i^*_{k-1}) + \breve{g}(k - i^*_{k-1}) + \delta,$$

where the first inequality follows due to the definition of P^-_δ and the second due to the definition of $\breve{h}$. Rearranging, we get

$$\breve{g}(k - i_k) - \breve{g}(k - i^*_{k-1}) \leq \breve{f}(i^*_{k-1}) - \breve{f}(i_k) + \delta. \tag{2.1}$$

Since $i^*_{k-1} \geq i_{k-1} \geq i_k$, we have $k - i_k \geq k - i^*_{k-1}$. By convexity of $\breve{g}$, we obtain

$$\breve{g}(k - 1 - i_k) - \breve{g}(k - 1 - i^*_{k-1}) \leq \breve{g}(k - i_k) - \breve{g}(k - i^*_{k-1}). \tag{2.2}$$

Combining (2.1) and (2.2) and rearranging, we obtain

$$\breve{f}(i_k) + \breve{g}(k - 1 - i_k) \leq \breve{f}(i^*_{k-1}) + \breve{g}(k - 1 - i^*_{k-1}) + \delta = \breve{h}(k-1) + \delta,$$

where the last equality is by definition of i^*_{k-1}. Thus, $(i_k, k - 1 - i_k)$ is δ-relevant, and since i_{k-1} is minimal such that $(i_{k-1}, k - 1 - i_{k-1})$ is δ-relevant we obtain $i_{k-1} \leq i_k$, as desired.

Case 2: $i_k > i_{k-1}$. We show that in this case $i_k \leq i_{k-1} + 1$ (and thus, $i_k = i_{k-1} + 1$). Let i^*_k be a witness for $\breve{h}(k)$. By definition of P^-_δ, we have $i^*_k \geq i_k$. Moreover,

$$\breve{f}(i_{k-1}) + \breve{g}(k - 1 - i_{k-1}) \leq \breve{h}(k-1) + \delta \leq \breve{f}(i^*_k - 1) + \breve{g}(k - i^*_k) + \delta,$$

where the first inequality is due to the definition of P^-_δ and the second due to the definition of $\breve{h}$. Rearranging, we get

$$\breve{g}(k - 1 - i_{k-1}) - \breve{g}(k - i^*_k) \leq \breve{f}(i^*_k - 1) - \breve{f}(i_{k-1}) + \delta. \tag{2.3}$$

Since $i^*_k - 1 \geq i_k - 1 \geq i_{k-1}$ and by the convexity of $\breve{f}$, we have

$$\breve{f}(i^*_k - 1) - \breve{f}(i_{k-1}) \leq \breve{f}(i^*_k) - \breve{f}(i_{k-1} + 1). \tag{2.4}$$

Combining and rearranging (2.3) and (2.4), we obtain

$$\breve{f}(i_{k-1} + 1) + \breve{g}(k - 1 - i_{k+1}) \leq \breve{f}(i^*_k) - \breve{g}(k - i^*_k) + \delta = \breve{h}(k) + \delta,$$

where the last equality holds by definition of i^*_k. Hence, $(i_{k-1} + 1, k - 1 - i_{k+1})$ is δ-relevant. Since its diagonal is k, $i_{k-1} + 1$ is a possible choice for i_k. By minimality of i_k (due to the definition of P^-_δ), we obtain that $i_k \leq i_{k-1} + 1$.

In both cases we obtain $i_k \in \{ i_{k-1}, i_{k-1} + 1 \}$, proving the claim. This finishes the proof for P^-_δ. The proof for P^+_δ is symmetric (replacing the roles of f and g essentially flips P^-_δ and P^+_δ). □

Let (i, j) be a point and P a monotone path. Let $(a, b) \in P$ be the unique point on the same diagonal as (i, j), i.e., $a + b = i + j$. We say that (i, j) is *below* P if $i < a$, *above* P if $i > a$, and *on* P if $i = a$, see Figure 2.1b for an illustration.

Lemma 2.4. *For any $\delta \geq 0$, R_δ consists of all points (i, j) that are on or below P^+_δ and on or above P^-_δ.*

Proof. Fix $k \in [n + m]$ and let $(i^+, k - i^+), (i^-, k - i^-)$ be the point on diagonal k in P^+_δ and P^-_δ, respectively. Consider any $(i, j) \in R_\delta$ on diagonal k. By maximality of i^+ we have $i \leq i^+$, and similarly $i \geq i^-$ by the minimality of i^-. Thus, no point in R_δ is above P^+_δ or below P^-_δ. It remains to show that for any $i^- \leq i \leq i^+$ we have $(i, k - i) \in R_\delta$. Note that the function $r(i) := \breve{f}(i) + \breve{g}(k - i)$ is convex (since it is the sum of convex functions). Since $(i^+, k - i^+)$ is δ-relevant, we have $r(i^+) \leq \breve{h}(k) + \delta$. Similarly, since $(i^-, k - i^-)$ is δ-relevant, we have $r(i^-) \leq \breve{h}(k) + \delta$. By convexity of r, we obtain that $r(i) \leq \breve{h}(k) + \delta$ for all $i^- \leq i \leq i^+$. Hence, we conclude that for each $i^- \leq i \leq i^+$ we have $(i, k - i) \in R_\delta$. □

Finally, we need some background on *sumsets*. Given $A, B \subseteq [-U \mathinner{..} U]^2$ where $U \in \mathbb{N}$, we define $A + B = \{ a + b \mid a \in A, b \in B \}$ as their sumset, where the addition $a + b$ is done componentwise. The naive way to compute $A + B$ takes time $O(|A| \cdot |B|)$. For our application, we want to compute the sumset in time near linear in its size $|A + B|$. For this end, we will use the following tool to compute *sparse nonnegative convolution*. Given vectors $P, Q \in \mathbb{N}^n$, their *convolution* $P \star Q \in \mathbb{N}^{2n-1}$ is defined coordinate-wise by $(P \star Q)[k] = \sum_{i+j=k} P[i] \cdot Q[j]$.

Theorem 2.5 (Deterministic Sparse Convolution [BFN22])**.** *There is a deterministic algorithm to compute the convolution of two nonnegative vectors $A, B \in \mathbb{N}^n$ in time $O(t \operatorname{polylog}(n\Delta))$, where t is the number of non-zero entries in $A \star B$ and Δ is the largest entry in A and B.*

See also [BFN21] for improvements in the log-factors at the cost of randomization and [CH02; Nak20; GGC20] for prior randomized algorithms with similar guarantees.

Corollary 2.6 (Output Sensitive Sumset Computation)**.** *Given $A, B \subseteq [-U \mathinner{..} U]^2$, with $|A + B| \leq N$, $A + B$ can be computed in time $\widetilde{O}(N)$.*

Proof. Let $A' := \{ (x+U) \cdot 5U + (y+U) \mid (x, y) \in A \}$ and similarly, let $B' := \{ (x+U) \cdot 5U + (y + U) \mid (x, y) \in B \}$. Observe that this is a one-to-one embedding of $A, B \subseteq [-U \mathinner{..} U]^2$ into $A', B' \subseteq [\Theta(U^2)]$. Moreover, one can check that given $C' := A' + B'$ we can infer $C := A + B$ (the choice of $5U$ prevents any interactions between coordinates when summing them up).

Thus, it suffices to compute $A' + B'$. To this end, construct their indicator vectors $P_{A'}, P_{B'} \in \mathbb{N}^{\Theta(U^2)}$ and compute the convolution $P_{C'} = P_{A'} \star P_{B'}$. The non-zero entries in $P_{C'}$ correspond to the elements of $A' + B'$. By Theorem 2.5, this runs in time $O(|A' + B'| \operatorname{polylog}(N, U)) = \widetilde{O}(N)$. □

2.2.2 Algorithm

We are ready to describe our algorithm. Recall that we have access to the functions $f, \breve{f}, g, \breve{g}$ and the value $\Delta = \max\{ \Delta_f, \Delta_g \}$.

Computing $\breve{h}$ = MinConv($\breve{f}, \breve{g}$). Consider the pseudocode given in Algorithm 1.

Algorithm 1 Given convex functions $\breve{f} \colon [n] \mapsto \mathbb{Q}, \breve{g} \colon [m] \mapsto \mathbb{Q}$, the algorithm computes $\breve{h} = \textsc{MinConv}(\breve{f}, \breve{g})$.

```
1  $i^*_0 \leftarrow 0, \breve{h}(0) \leftarrow \breve{f}(0) + \breve{g}(0)$
2  for $k = 1, \dots, n+m$ do
3      $i^*_k \leftarrow \operatorname{argmin}\{ \breve{f}(i) + \breve{g}(k-i) + \frac{i}{2n} \mid i \in \{ i^*_{k-1}, i^*_{k-1} + 1 \} \cap [n] \}$
4      $\breve{h}(k) \leftarrow \breve{f}(i^*_k) + \breve{g}(k - i^*_k)$
```

Lemma 2.7. *Algorithm 1 computes $\breve{h}$ = MinConv($\breve{f}, \breve{g}$) in time $O(n+m)$.*

Proof. The running time is immediate. To see correctness, focus on i^*_k for $k \in [n+m]$ as computed in Algorithm 1. We claim that the path P^-_0 equals $\{ (i^*_k, k - i^*_k) \mid k \in [n+m] \}$. That is, we want to argue that i^*_k is the minimum witness of $\breve{h}(k)$ for each $k \in [n+m]$. Indeed, by Lemma 2.3, P^-_0 is a monotone path. Thus, $i^*_k \in \{ i^*_{k-1}, i^*_{k-1} + 1 \}$. Observe that in Line 3 we pick i^*_k as the minimizer of $\breve{f}(i) + \breve{g}(k-i) + \frac{i}{2n}$ where $i \in \{ i^*_{k-1}, i^*_{k-1} + 1 \}$. Therefore, the algorithm correctly computes i^*_k (the additive term $i/(2n)$ ensures that we choose the minimal i). Since i^*_k is a minimum witness of $\breve{h}(k)$, the algorithm correctly computes $\breve{h}(k)$ for all $k \in [n+m]$. □

Computing h = MinConv(f, g). Recall that $f \colon [n] \mapsto \mathbb{Z}$ and $g \colon [m] \mapsto \mathbb{Z}$. As a final simplification, we argue that we can assume without loss of generality that $n = m$, and $n + 1$ is a power of 2. To this end, let N be the smallest power of 2 greater than $\max\{ n, m \}$. We pad the functions to length N by setting $f(n + j) := 2j \cdot W$ for $j \in [1 \,..\, N-1-n]$ and $g(m+j) := 2j \cdot W$ for $j \in [1 \,..\, N-1-m]$, where W is an integer larger than $\max_{i \in [n]} f(i) + \max_{j \in [m]} g(j)$. Observe that the entries $h(0), \dots, h(n+m)$ of the result $h = \textsc{MinConv}(f, g)$ are unchanged (due to the choice of sufficiently large W), so we can read off the original result from the result of the padded functions. Moreover, observe that the padding does not change the parameters Δ_f and Δ_g.

Now we can describe the algorithm. After running Algorithm 1 we can assume that we have computed $\check{h}$ and the witness path $P_0^- = \{ (i_k^*, k - i_k^*) \mid k \in [n+m] \}$. We will make use of the following boolean subroutines:

- $\textsc{Relevant}(i, j)$: returns $\check{f}(i) + \check{g}(j) \leq \check{h}(i+j) + 2\Delta$.
- $\textsc{BelowWitnessPath}(i, j)$: returns $i < i_{i+j}^*$
- $\textsc{AboveWitnessPath}(i, j)$: returns $i > i_{i+j}^*$

Now we can compute $h = \textsc{MinConv}(f, g,)$ by calling $\textsc{RecMinConv}([0\,..\,n], [0\,..\,m])$. See Algorithm 2 for the pseudocode.

Algorithm 2 Given intervals $I = [\,i_A\,..\,i_B\,], J = [\,j_A\,..\,j_B\,]$, the algorithm computes the contribution of $f[I]$ and $g[J]$ to $\textsc{MinConv}(f, g)$.

```
procedure RecMinConv($I = [\,i_A\,..\,i_B\,], J = [\,j_A\,..\,j_B\,]$)
    if AboveWitnessPath($i_A, j_B$) and NotRelevant($i_A, j_B$) then          ▷ Case 1
        return $\tilde{h}(k) = \infty$ for all $k \in [i_A + j_A\,..\,i_B + j_B]$
    if BelowWitnessPath($i_A, j_B$) and NotRelevant($i_B, j_A$) then          ▷ Case 2
        return $\tilde{h}(k) = \infty$ for all $k \in [i_A + j_A\,..\,i_B + j_B]$
    if Relevant($i_A, j_B$) and Relevant($i_B, j_A$) then                     ▷ Case 3
        Compute $C \leftarrow \{ (i, f(i)) \mid i \in I \} + \{ (j, g(j)) \mid j \in J \}$ using Corollary 2.6
        Infer $\tilde{h}(k) \leftarrow \min\{ y \mid (k, y) \in C \}$ for all $k \in [i_A + j_A\,..\,i_B + j_B]$
        return $\tilde{h}$
    else                                                                      ▷ Case 4
        Split $I$ into two intervals $I_1, I_2$ of equal length, similarly split $J$ into $J_1, J_2$
        Recursively compute $\tilde{g}_{i,j} \leftarrow$ RecMinConv($I_i, J_j$) for $i, j \in \{1, 2\}$
        return the pointwise minimum of the functions $\tilde{g}_{i,j}$ for $i, j \in \{1, 2\}$
```

Algorithm 2 recursively computes the contribution of $f[i_A\,..\,i_B]$ and $g[j_A\,..\,j_B]$ to $h = \textsc{MinConv}(f, g)$. We next discuss its four cases; see Figure 2.2 for illustrations of Cases 1-3. If (i_A, j_B) is above the witness path P_0^- and is not 2Δ-relevant (Case 1), then as we argue below no point in $I \times J$ contributes to the output h, so in this case we return a dummy function (which is $+\infty$ everywhere). Case 2 is symmetric, where (i_B, j_A) is above P_0^- and not 2Δ-relevant, and we again return a dummy function. Case 3 applies when (i_A, j_B) and (i_B, j_A) are both 2Δ-relevant. In this case, we explicitly compute $\tilde{h} = \textsc{MinConv}(f[i_A\,..\,i_B], g[j_A\,..\,j_B])$ by computing the sumset $C = \{ (i, f(i)) \mid i \in I \} + \{ (j, g(j)) \mid j \in J \}$ and inferring $\tilde{h}(k)$ as the minimum y such that $(k, y) \in C$, which by definition of the sumset equals the minimum $f(i) + g(j)$ such that $i \in I, j \in J$ and $i + j = k$. Note that this step can be done for all $k \in [i_A + j_A\,..\,i_B + j_B]$ in total time $O(|C|)$ by once scanning over all elements of C.

Finally, if none of the above cases apply, then we split both intervals I and J into equal halves and recurse on all 4 combinations of halves. We combine them by taking the pointwise minimum of all computed functions. This case is essentially brute force.

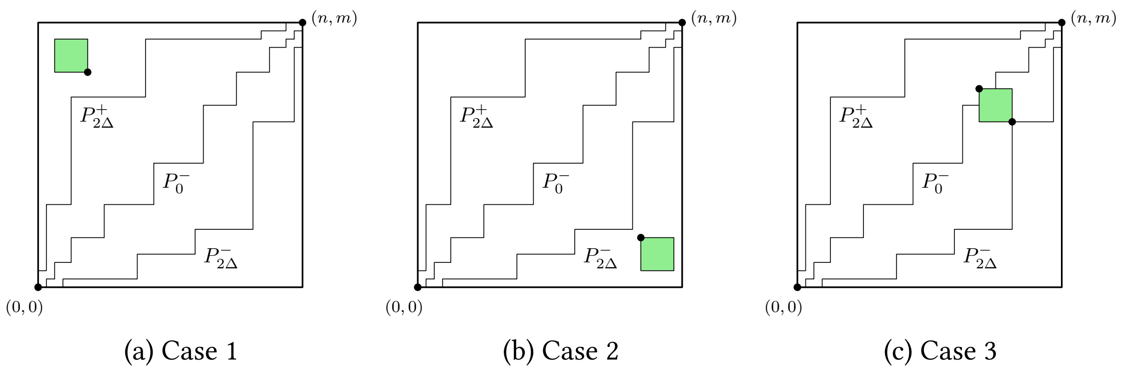


Figure 2.2: Visualization of Cases 1-3 of Algorithm 2. The green box represents the current subproblem.

Correctness

We start by analyzing the correctness of the algorithm.

Lemma 2.8 (Correctness of Algorithm 2)**.** *RecMinConv$([0\,..\,n], [0\,..\,m])$ (Algorithm 2) correctly computes $h = $ MinConv(f, g).*

Proof. Let $k \in [n+m]$ and consider a point (i^*, j^*) in diagonal k such that $f(i^*)+g(j^*) = h(k)$, i.e., a witness for $h(k)$. We argue that some recursive call computes $f(i^*) + g(j^*)$. This is clear in Case 4, as (i^*, j^*) is covered by one recursive subproblem. It is also clear in Case 3, since then $f(i^*) + g(j^*)$ is explicitly computed.

To finish correctness, we argue that (i^*, j^*) can never be in a subproblem to which Case 1 or 2 applies. Recall that Case 1 applies to a subproblem $I = [i_A\,..\,i_B], J = [j_A\,..\,j_B]$ if (i_A, j_B) is above P^-_0 and (i_A, j_B) is not 2Δ-relevant. Since (i_A, j_B) is not 2Δ-relevant, by Lemma 2.4 (i_A, j_B) must be above $P^+_{2\Delta}$ or below $P^-_{2\Delta}$. Since (i_A, j_B) is above P^-_0, it can only be above $P^+_{2\Delta}$. Furthermore, since (i_A, j_B) is the lower right corner of $I \times J$, it follows that all points in $I \times J$ are above $P^+_{2\Delta}$. Thus, by Lemma 2.4 all points in $I \times J$ are not 2Δ-relevant. If we assume for the sake of contradiction that $(i^*, j^*) \in I \times J$, then Lemma 2.2 implies $f(i^*) + g(j^*) > h(k)$, contradicting the choice of (i^*, j^*) as a witness for $h(k)$. Hence, (i^*, j^*) can never be in a Case 1 subproblem. Case 2 is symmetric. This finishes the correctness proof. □

Running Time

Next, we analyze the running time. The key insight is that in relevant regions both functions are essentially linear, with the same slope (see Lemma 2.9). This implies that the sumset computed in Case 3 is small (see Lemma 2.10), so it can be computed efficiently using Corollary 2.6. In the following two lemmas, let $I = [i_A\,..\,i_B] \subseteq [n]$ and $J = [j_A\,..\,j_B] \subseteq [m]$ be intervals of the same length $|I| = |J|$.

Lemma 2.9 (Near Linearity inside Relevant Region). *If $I \times J \subseteq R_{2\Delta}$ then there are $a, b, c \in \mathbb{R}$ such that $|f(i) - (a \cdot i + b)| \le 2\Delta$ for all $i \in I$ and $|g(j) - (a \cdot j + c)| \le 2\Delta$ for all $j \in J$.*

Proof. Consider the linear interpolation between $(i_A, \breve{f}(i_A))$ and $(i_B, \breve{f}(i_B))$:

$$F(x) := \frac{(i_B - x)\breve{f}(i_A) + (x - i_A)\breve{f}(i_B)}{i_B - i_A}.$$

Similarly, consider

$$G(x) := \frac{(j_B - x)\breve{g}(j_A) + (x - j_A)\breve{g}(j_B)}{j_B - j_A}.$$

By convexity of $\breve{f}$ and $\breve{g}$, we have

$$\breve{f}(i) \le F(i) \quad \forall i \in I, \qquad \breve{g}(j) \le G(j) \quad \forall j \in J. \tag{2.5}$$

Consider the diagonal $k := i_A + j_B$ and note that for all $i \in I$ we have $k - i \in J$ due to $|I| = |J|$. Thus, for each $i \in I$ the point $(i, k - i)$ is 2Δ-relevant, and we obtain

$$\breve{h}(k) \le \breve{f}(i) + \breve{g}(k-i) \le \breve{h}(k) + 2\Delta \quad \forall i \in I \tag{2.6}$$

This implies

$$\breve{h}(k) \le F(i) + G(k-i) \le \breve{h}(k) + 2\Delta \quad \forall i \in I, \tag{2.7}$$

since by (2.6) these inequalities hold for $i \in \{ i_A, i_B \}$ and by the linear interpolation, they also hold in between.

Now for any $i \in I$ we have

$$\breve{f}(i) \overset{(2.6)}{\ge} \breve{h}(k) - \breve{g}(k-i) \overset{(2.5)}{\ge} \breve{h}(k) - G(k-i) \overset{(2.7)}{\ge} \breve{h}(k) - (\breve{h}(k) + 2\Delta - F(i)) = F(i) - 2\Delta.$$

Thus, $\breve{f}(i) \in [F(i) - 2\Delta \,..\, F(i)]$, and by $\breve{f} \le f \le \breve{f} + \Delta_f \le \breve{f} + \Delta$, we obtain that $|f(i) - F(i)| \le 2\Delta$.

For $\breve{g}(j)$ for any $j \in J$ we bound

$$\breve{g}(j) \overset{(2.6)}{\ge} \breve{h}(k) - \breve{f}(k-j) \overset{(2.5)}{\ge} \breve{h}(k) - F(k-j),$$

and

$$\breve{g}(j) \overset{(2.5)}{\le} G(j) \overset{(2.7)}{\le} \breve{h}(k) - F(k-j) + 2\Delta.$$

Therefore, $|\breve{g}(j) - (\breve{h}(k) - F(k-j) + \Delta)| \le \Delta$. By linearity of F we can write $F(k-j) = F(k) - F(j) + F(0)$. This yields $|\breve{g}(j) - (F(j) + \lambda)| \le \Delta$ for $\lambda := \breve{h}(k) - F(k) - F(0) + \Delta$. Since $|g(j) - \breve{g}(j)| \le \Delta_g \le \Delta$ we obtain $|g(j) - (F(j) + \lambda)| \le 2\Delta$. Since F is linear, writing $F(i) = a \cdot i + b$ and $F(j) + \lambda = a \cdot j + c$ finishes the proof. □

Lemma 2.10 (Relevant Regions have Small Sumsets). *If $I \times J \subseteq R_{2\Delta}$ then the sumset $\{ (i, f(i)) \mid i \in I \} + \{ (j, f(j)) \mid j \in J \}$ has size $O(\Delta \cdot (|I| + |J|))$.*

Proof. By Lemma 2.9, for any $(i, j) \in I \times J$ with $i + j = k$ we have

$$f(i) + g(j) = (a \cdot i + b) + (a \cdot j + c) \pm O(\Delta) = a \cdot k + b + c \pm O(\Delta).$$

Thus, for each of the $|I| + |J| - 1$ x-coordinates (i.e., choices of $i + j$), there are $O(\Delta)$ different y-coordinates (i.e., values $f(i) + g(j)$) in the sumset. □

Lemma 2.11 (Running Time of Algorithm 2)**.** *RecMinConv*$([0\,..\,n], [0\,..\,m])$ *(Algorithm 2) runs in time $\widetilde{O}(n\Delta)$.*

Proof. We first analyze the running time of one recursive subproblem, ignoring the cost of recursive calls. Note that in Cases 1 and 2 it suffices to return a dummy value, i.e., we do not need to iterate over $k \in [i_A + j_A\,..\,i_B + j_B]$ to explicitly return $\tilde{h}(k) = \infty$. Thus, Cases 1 and 2 run in time $O(1)$. We charge this time to the parent of the current subproblem, which is a Case 4-subproblem.

Consider Case 4. Ignoring the cost of the recursive subproblems, Case 4 runs in time $O(1)$, which also covers the charging from children which fall in Cases 1 and 2.

Consider Case 3, and let $s := i_B - i_A + 1 = j_B - j_A + 1$ be the current side length. By Lemma 2.10, the sumset computed in Line 7 has size $O(\Delta s)$. Thus, it can be computed in time $\widetilde{O}(\Delta s)$ using Corollary 2.6, and the function $\tilde{h}$ can be inferred from it in time $O(\Delta s)$.

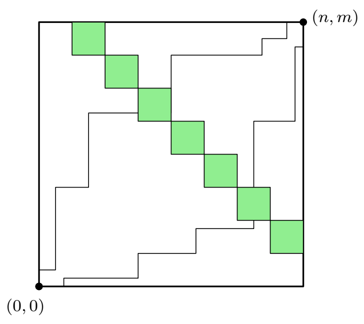
Now we bound the total running time across subproblems. Fix a side length s and consider all possible subproblems of side length s, i.e., all boxes

$$B^s_{x,y} := [x \cdot s\,..\,x \cdot s + s - 1] \times [y \cdot s\,..\,y \cdot s + s - 1], \text{ where } x, y \in [n/s].$$

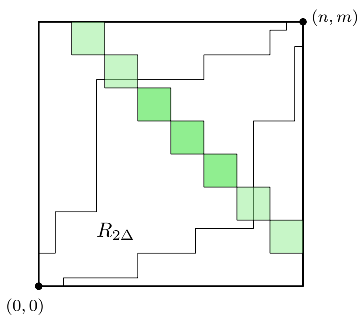
Consider a diagonal $D_{s,d} := \{\, B^s_{x,x+d} \mid x \in [n/s] \,\}$ of these boxes, see Figure 2.3a. Note that a box in $D_{s,d}$ that lies fully above P^+_Δ corresponds to a Case 1-subproblem. A box in $D_{s,d}$ that lies fully below corresponds to a Case 2-subproblem. A box that is below or on $P^+_{2\Delta}$ and above or on $P^-_{2\Delta}$ corresponds to a Case 3-subproblem. The remaining boxes intersect $P^+_{2\Delta}$ or $P^-_{2\Delta}$ and correspond to Case 4.

Note that by monotonicity of $P^+_{2\Delta}, P^-_{2\Delta}$, at most two boxes in $D_{s,d}$ are intersected by $P^+_{2\Delta}$ or $P^-_{2\Delta}$ and thus at most two boxes in $D_{s,d}$ can appear as Case 4-subproblems. Thus, Case 4 incurs time $O(1)$ per diagonal. We argue that among the boxes in $D_{s,d}$, at most two can appear as Case 3-subproblems. Indeed, if these would be at least three such boxes, then the parent of the middle box would also be between $P^+_{2\Delta}$ and $P^-_{2\Delta}$, and thus the parent would already be a Case 3-subproblem, see Figures 2.3b and 2.3c. Thus, the middle box would not get split, and it would not become a recursive subproblem. Hence, per diagonal $D_{s,d}$, Case 3 incurs time $\widetilde{O}(\Delta s)$ for each of at most two boxes.

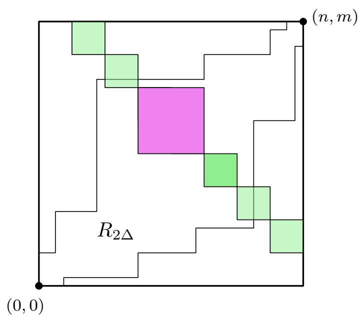
It remains to sum up over all side lengths $1 \le s \le n$ where $s = 2^\ell$ is a power of 2 (recall that at each recursive level we split the side length in two equal parts), and over all $O(n/s)$ diagonals d, to obtain total time $\sum_{\ell=1}^{\log n} O(n/2^\ell) \cdot \widetilde{O}(\Delta 2^\ell) = \widetilde{O}(\Delta n)$. Note that the sum over ℓ only adds another log-factor, which is hidden by the $\widetilde{O}$-notation. □



(a) A diagonal of boxes $D_{s,d}$



(b) Three boxes inside $R_{2\Delta}$



(c) The parent box is already contained in $R_{2\Delta}$

Figure 2.3: Visualizations for the proof of Lemma 2.11.

2.3 Faster Knapsack Algorithm

In this section we prove Main Theorem 1.2, restated for convenience.

Main Theorem 1.2. *There is a randomized algorithm for Knapsack that runs in time*

$$\widetilde{O}((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max})$$

and succeeds with high probability. Using the bound $W \le nw_{\max}$*, this running time is at most* $\widetilde{O}(nw_{\max}\, p_{\max}^{2/3})$.

Let $(\mathcal{I}, W)$ be a Knapsack instance. Throughout, we denote the number of items by $n := |\mathcal{I}|$. We identify the item set $\mathcal{I}$ with $\{1,\dots,n\}$. We denote by $p_{\max} := \max_i p_i$ the maximum profit and by $w_{\max} := \max_i w_i$ the maximum weight in $\mathcal{I}$. We represent a *solution* to the knapsack instance (i.e., a subset of $\mathcal{I}$), by an indicator vector $x \in \{0,1\}^n$. For a subset of the items $\mathcal{J} \subseteq \mathcal{I}$, we put $w_{\mathcal{J}}(x) := \sum_{i\in\mathcal{J}} w_i x_i$ and $p_{\mathcal{J}}(x) := \sum_{i\in\mathcal{J}} p_i x_i$. We define the profit sequence $\mathcal{P}_{\mathcal{I}}[\cdot]$, where for each $j \in \mathbb{N}$ we have

$$\mathcal{P}_{\mathcal{I}}[j] = \max\{\, p_{\mathcal{I}}(x) \mid x \in \{0,1\}^n, w_{\mathcal{I}}(x) \le j \,\}.$$

Observe that $\mathcal{P}_{\mathcal{I}}$ is monotone non-decreasing, and that $\mathrm{OPT} = \mathcal{P}_{\mathcal{I}}[W]$. The textbook way to compute $\mathcal{P}_{\mathcal{I}}[0\,..\,j]$ is to use dynamic programming:

Fact 2.12. *For any* $j \in \mathbb{N}$ *the sequence* $\mathcal{P}_{\mathcal{I}}[0\,..\,j]$ *can be computed in time* $O(nj)$.

Before presenting the algorithm, we make two simple observations about the given Knapsack instance $(\mathcal{I}, W)$. First, by ignoring items with weight larger than the capacity W, we can assume without loss of generality that $w_{\max} \le W$. Now every single item is a feasible solution, so we have $p_{\max} \le \mathrm{OPT}$. Second, observe that if $W \ge n \cdot w_{\max}$, then the instance is trivial since we can pack all items. Thus, we can assume without loss of generality that $W \le n \cdot w_{\max}$. Moreover, since any feasible solution consists of at most all the n items, it follows that $\mathrm{OPT} \le n \cdot p_{\max}$.

The Algorithm

We now describe the algorithm. We set the parameters

$$q := \min\{\, (n/p_{\max})^{2/3}(W/w_{\max})^{1/3}, W/w_{\max}\,\}$$

rounded down to the closest power of 2, $\Delta := w_{\max}W/q$ and $\eta := 11\log n$. For each $\ell \in [\log q]$ we define the interval $J^\ell := [\frac{W}{q}2^\ell - \sqrt{\Delta 2^\ell}\eta \,.\,.\, \frac{W}{q}2^\ell + \sqrt{\Delta 2^\ell}\eta]$.

We start by splitting the items $\mathcal{I}$ into q groups $\mathcal{I}_1^0, \dots, \mathcal{I}_q^0$ uniformly at random. The idea will be to compute an array C_j^0 associated to each $\mathcal{I}_j^0$, and then combine them in a *tree-like* fashion. A crucial aspect for the running time is that we only compute $|J^\ell|$ entries of each array C_j^ℓ. In detail, we proceed as follows:

Base Case For each $\mathcal{I}_j^0$, we use Fact 2.12 to compute $\mathcal{P}_{\mathcal{I}_j^0}[0\,.\,.\,\frac{W}{q} + \sqrt{\Delta}\eta]$ and define the subarray $C_j^0[J^0] := \mathcal{P}_{\mathcal{I}_j^0}[J^0]$.

Combination Iterate over the *levels* $\ell = 1, \dots q$. For $j \in [1\,.\,.\,q/2^\ell]$ we set $\mathcal{I}_j^\ell := \mathcal{I}_{2j-1}^{\ell-1} \cup \mathcal{I}_{2j}^{\ell-1}$. Then, compute the subarray $C_j^\ell[J^\ell]$ by taking the relevant entries of the max-plus convolution of $C_{2j-1}^{\ell-1}[J^{\ell-1}]$ and $C_{2j}^{\ell-1}[J^{\ell-1}]$.

Returning the answer (Note that when $\ell = \log(q)$, it holds that $\mathcal{I}_1^{\log q} = \mathcal{I}$.) We return the value $C_1^{\log q}[W]$.

Refer to Algorithm 3 for the complete pseudocode.

Correctness

We start by analyzing the correctness of the algorithm. The following lemma shows that the weight of any solution restricted to one of the sets $\mathcal{I}_j^\ell$ is concentrated around its expectation.

Lemma 2.13 (Concentration). *Let $x \in \{0, 1\}^n$ be a solution to the given Knapsack instance. Fix a level $\ell \in [0\,.\,.\,\log q]$ and $j \in [1\,.\,.\,q/2^\ell]$. Then, with probability at least $1 - 1/n^4$ it holds that:*

$$\left| w_{\mathcal{I}_j^\ell}(x) - \frac{w_{\mathcal{I}}(x)\cdot 2^\ell}{q} \right| \le \sqrt{\Delta 2^\ell}\cdot 10\log n.$$

Proof. Recall that the item set $\mathcal{I}$ is partitioned randomly into $\mathcal{I}_1^0, \dots, \mathcal{I}_q^0$. Thus, observe that $\mathcal{I}_j^\ell$ is a random subset of $\mathcal{I}$, where each item is included with probability $p := 2^\ell/q$. For $i \in [1\,.\,.\,n]$, let Z_i be a random variable which equals $w_i \cdot x_i$ with probability p, and 0 with probability $1 - p$. Then, observe that $w_{\mathcal{I}_j^\ell}(x)$ is distributed as $Z := \sum_{i=1}^n Z_i$, and therefore, $\mathbb{E}(Z) = w_{\mathcal{I}}(x)p$.

Algorithm 3 Knapsack Algorithm. Given a set of items $\mathcal{I}$ and a weight budget W, the algorithm computes the maximum attainable profit.

```
1  $q \leftarrow \min\{ (n/p_{\max})^{2/3}(W/w_{\max})^{1/3}, W/w_{\max} \}$ rounded down to the closest power
   of 2
2  $\Delta \leftarrow w_{\max}W/q$
3  $\eta \leftarrow 11 \log n$
4  $\mathcal{I}_1^0, \dots, \mathcal{I}_q^0 \leftarrow$ random partitioning of $\mathcal{I}$ into $q$ groups
5  for $i = 1 \dots q$ do
6      Compute $\mathcal{P}_{\mathcal{I}_j^0}[0 \,..\, \frac{W}{q} + \sqrt{\Delta}\eta]$ using standard dynamic programming (Fact 2.12)
7      $J^0 \leftarrow [\frac{W}{q} - \sqrt{\Delta}\eta \,..\, \frac{W}{q} + \sqrt{\Delta}\eta]$
8      $C_j^0[J^0] \leftarrow \mathcal{P}_{\mathcal{I}_j^0}[J^0]$
9  for $\ell = 1 \dots \log(q)$ do
10     $J^\ell \leftarrow [\frac{W}{q}2^\ell - \sqrt{\Delta 2^\ell}\eta \,..\, \frac{W}{q}2^\ell + \sqrt{\Delta 2^\ell}\eta]$
11     for $j = 1, \dots, q/2^\ell$ do
12         $\mathcal{I}_j^\ell \leftarrow \mathcal{I}_{2j-1}^{\ell-1} \cup \mathcal{I}_{2j}^{\ell-1}$
13         Compute $C_j^\ell[J^\ell] \leftarrow \textsc{MaxConv}(C_{2j-1}^{\ell-1}[J^{\ell-1}], C_{2j}^{\ell-1}[J^{\ell-1}])$ using
               Main Theorem 1.7
14 return $C_1^{\log q}[W]$
```

To prove the statement, we will use Bernstein's inequality (see e.g. [DP09, Theorem 1.2]) which states that

$$\begin{aligned}\mathbb{P}(|Z - \mathbb{E}(Z)| \geq t) &\leq 2\exp\left(-\frac{t^2}{2\,\mathbf{Var}(Z) + \frac{2}{3}t \cdot w_{\max}}\right)\\ &\leq 2\exp\left(-\min\left\{\frac{t^2}{4\,\mathbf{Var}(Z)}, \frac{t}{2w_{\max}}\right\}\right). \qquad (2.8)\end{aligned}$$

Set $t := \sqrt{p \cdot w_{\max}W} \cdot 10 \log n$. We first bound $t^2/(4\,\mathbf{Var}(Z))$. Note that we can give an upper bound on the variance as follows:

$$\mathbf{Var}(Z) = \sum_{i=1}^{n} p(1-p)w_i^2x_i^2 \leq p \cdot w_{\max} \sum_{i=1}^{n} w_ix_i = p \cdot w_{\max}w_{\mathcal{I}}(x) \leq p \cdot w_{\max}W.$$

Therefore, $t^2/(4\,\mathbf{Var}(Z)) \geq 10 \log n$. Next, we bound $t/(2w_{\max})$. Using that $q \leq W/w_{\max}$, we have that $p = \frac{2^\ell}{q} \geq \frac{w_{\max}2^\ell}{W} \geq \frac{w_{\max}}{W}$. Thus,

$$\frac{t}{2w_{\max}} = \frac{\sqrt{p \cdot w_{\max}W} \cdot 10 \log n}{2w_{\max}} \geq 5 \log n.$$

Combining the above, we obtain from (2.8) that

$$|w_{\mathcal{I}_j^\ell}(x) - w_{\mathcal{I}}(x)2^\ell/q| = |Z - \mathbb{E}(Z)| \leq t = \sqrt{pw_{\max}W} \cdot 10 \log n = \sqrt{\Delta 2^\ell} \cdot 10 \log n$$

holds with probability at least $1 - 2/n^5 \geq 1 - 1/n^4$. □

Using Lemma 2.13, we can argue that at level ℓ it suffices to compute a subarray of length $\widetilde{O}(\sqrt{\Delta 2^\ell})$ around $W2^\ell/q$. The following lemma makes this precise:

Lemma 2.14. *Let $x \in \{0,1\}^n$ be a solution to the given Knapsack instance satisfying $w_{\mathcal{I}}(x) \in [W - w_{\max} \,..\, W]$. With probability at least $1 - 1/n^2$, for all levels $\ell \in [0 \,..\, \log q]$ and all $j \in [1 \,..\, q/2^\ell]$ it holds that:*

- $w_{\mathcal{I}_j^\ell}(x) \in J^\ell = [\frac{W}{q}2^\ell - \sqrt{\Delta 2^\ell}\eta \,..\, \frac{W}{q}2^\ell + \sqrt{\Delta 2^\ell}\eta]$*, and*
- $C_j^\ell[w_{\mathcal{I}_j^\ell}(x)] \geq p_{\mathcal{I}_j^\ell}(x)$.

Proof. By Lemma 2.13, for each $\ell \in [0 \,..\, \log q]$ and $j \in [1 \,..\, q/2^\ell]$ it holds that

$$|w_{\mathcal{I}_j^\ell}(x) - w_{\mathcal{I}}(x)2^\ell/q| \leq \sqrt{\Delta 2^\ell} \cdot 10 \log n \tag{2.9}$$

with probability at least $1 - 1/n^4$. Note that $q \leq W/w_{\max} \leq n$. Thus, we can afford a union bound and conclude that (2.9) holds *for all* $\ell \in [0 \,..\, \log q]$ and $j \in [1 \,..\, q/2^\ell]$ with probability at least $1 - 1/n^2$. From now on, we condition on this event.

We start by showing the first item of the statement. Fix $\ell \in [\log q]$ and $j \in [1 \,..\, q/2^\ell]$. By (2.9), it holds that $|w_{\mathcal{I}_j^\ell}(x) - w_{\mathcal{I}}(x)2^\ell/q| \leq \sqrt{\Delta 2^\ell} \cdot 10 \log n$. By assumption, we have that $w_{\mathcal{I}}(x) \in [W - w_{\max} \,..\, W]$. Hence,

$$\begin{aligned} |w_{\mathcal{I}_j^\ell}(x) - W2^\ell/q| &\leq |w_{\mathcal{I}_j^\ell}(x) - w_{\mathcal{I}}(x)2^\ell/q| + \tfrac{2^\ell}{q}|w_{\mathcal{I}}(x) - W| \\ &\leq \sqrt{\Delta 2^\ell} 10 \log n + w_{\max}2^\ell/q \leq \sqrt{\Delta 2^\ell} \cdot 11 \log n. \end{aligned}$$

The last inequality holds since we can use that $2^\ell \leq q$ and $w_{\max} \leq W$ to obtain that $w_{\max}2^\ell/q \leq \sqrt{w_{\max}W} \cdot \sqrt{2^\ell/q} = \sqrt{\Delta 2^\ell}$. Since $\eta = 11 \log n$, this implies that $w_{\mathcal{I}_j^\ell}(x) \in J^\ell = [\frac{W}{q}2^\ell - \sqrt{\Delta 2^\ell}\eta \,..\, \frac{W}{q}2^\ell + \sqrt{\Delta 2^\ell}\eta]$. This concludes the proof of the first item.

Next, we prove the second item of the lemma by induction. Consider the base case $\ell = 0$. By the first item, for any $j \in [1 \,..\, q]$ we have that $w_{\mathcal{I}_j^0}(x) \in J^0$. In particular, it holds that $C_j^0[w_{\mathcal{I}_j^0}(x)] = \mathcal{P}_{\mathcal{I}_j^0}[w_{\mathcal{I}_j^0}(x)]$ (see Line 8). Then, since $\mathcal{P}_{\mathcal{I}_j^0}[i]$ is the maximum profit of a subset of items from $\mathcal{I}_j^0$ of weight at most i, it holds that $\mathcal{P}_{\mathcal{I}_j^0}[w_{\mathcal{I}_j^0}(x)] \geq p_{\mathcal{I}_j^0}(x)$, which completes the proof of the base case.

Now we proceed with the inductive step: Fix $\ell \geq 1$ and assume that $C_j^{\ell-1}[w_{\mathcal{I}_j^{\ell-1}}(x)] \geq p_{\mathcal{I}_j^{\ell-1}}(x)$ hold for all $j \in [1 \,..\, q/2^{\ell-1}]$. By the first item of the lemma, for each $j \in [1 \,..\, q/2^\ell]$ we have that $w_{\mathcal{I}_j^\ell}(x) \in J^\ell$. Thus, by the computation of Line 13, it holds that

$$\begin{aligned} C_j^\ell[w_{\mathcal{I}_j^\ell}(x)] &= \max\{\, C_{2j-1}^{\ell-1}[i] + C_{2j}^\ell[i'] \mid i, i' \in J^{\ell-1},\ i + i' = w_{\mathcal{I}_j^\ell}(x)\,\} \\ &\geq C_{2j-1}^{\ell-1}[w_{\mathcal{I}_{2j-1}^{\ell-1}}(x)] + C_{2j}^\ell[w_{\mathcal{I}_{2j}^{\ell-1}}(x)] \\ &\geq p_{\mathcal{I}_{2j-1}^{\ell-1}}(x) + p_{\mathcal{I}_{2j}^{\ell-1}}(x) = p_{\mathcal{I}_j^\ell}(x). \end{aligned}$$

In the second step, we used that $w_{\mathcal{I}_{2j-1}^{\ell-1}}(x), w_{\mathcal{I}_{2j}^{\ell-1}}(x) \in J^{\ell-1}$ as shown earlier. The third step follows from the induction hypothesis. The last equality holds since $\mathcal{I}_j^{\ell} = \mathcal{I}_{2j-1}^{\ell-1} \cup \mathcal{I}_{2j}^{\ell-1}$. ◻

Lemma 2.15 (Correctness of Algorithm 3)**.** *Let $x^* \in \{0,1\}^n$ be an optimal solution to the given Knapsack instance. Then, for every $i \in [w_{\mathcal{I}}(x^*) \,..\, W]$, it holds that $C_1^{\log q}[i] = \mathcal{P}_{\mathcal{I}}[i]$ with probability at least $1 - 1/n^2$.*

Proof. We can check in linear time $O(n)$ whether the optimal solution consists of all items, in which case the instance is trivial. Thus, we can assume without loss of generality that x^* does not include all items. In particular, x^* leaves at least one item out and therefore its weight satisfies $w_{\mathcal{I}}(x^*) \in [W - w_{\max} \,..\, W]$. By Lemma 2.14, it holds that $C_1^{\log q}[w_{\mathcal{I}}(x^*)] \geq p_{\mathcal{I}}(x^*) = \mathcal{P}_{\mathcal{I}}[w_{\mathcal{I}}(x^*)]$ with probability at least $1 - 1/n^2$. From now on we condition on this event. We will use the following auxiliary claim:

▷ Claim 2.16. The sequence $C_1^{\log q}[J^{\log q}]$ is monotone non-decreasing, and satisfies $C_1^{\log q}[i] \leq \mathcal{P}_{\mathcal{I}}[i]$ for all $i \in J^{\log q}$.

Proof. First we argue monotonicity by induction. Note that in the base case $\ell = 0$, the sequence $C_j^0[J^0] = \mathcal{P}_{\mathcal{I}_j^0}[J^0]$ is monotone non-decreasing due to the definition of $\mathcal{P}_{\mathcal{I}_j^0}$. For level $\ell > 0$, the sequence C_j^{ℓ} is computed by taking the max-plus convolution of sequences of level $\ell - 1$. The result follows by observing that the max-plus convolution of two monotone non-decreasing sequences is monotone non-decreasing.

The second part of the claim follows since (inductively) every entry $C_1^{\log q}[i]$ for $i \in J^{\log q}$ corresponds to the profit of a subset of items of $\mathcal{I}$ of weight at most i. ◁

Since x^* is an optimal solution, it holds that $\mathcal{P}_{\mathcal{I}}[i] = p_{\mathcal{I}}(x^*)$ for all $i \in [w_{\mathcal{I}}(x^*) \,..\, W]$. Thus, Claim 2.16 yields that $C_1^{\log q}[i] = \mathcal{P}_{\mathcal{I}}[i]$ for all $i \in [w_{\mathcal{I}}(x^*) \,..\, W]$, completing the proof. ◻

Running Time

Now we analyze the running time of Algorithm 3. The key speedup comes from the computation in Line 13, where we use Main Theorem 1.7 to perform the max-plus convolution. Since Main Theorem 1.7 is phrased in terms of min-plus convolution of near-convex functions, we will use the following corollary:

Corollary 2.17. *Let $f\colon [n] \mapsto [-U \,..\, U]$ and $g\colon [m] \mapsto [-U \,..\, U]$ be given as inputs, where $U \in \mathbb{N}$. Let $\Delta \geq 1$ be such that both f and g are Δ-near concave. Then, $\textsc{MaxConv}(f, g)$ can be computed in time $\widetilde{O}((n+m)\Delta)$*

Proof. Noting that $-f$ and $-g$ are Δ-near convex (Definition 2.1), the result follows from Main Theorem 1.7 and Fact 1.10. ◻

The following lemma shows that the max-plus convolution of two near-concave sequences remains near-concave.

Lemma 2.18. *Let $f\colon [n] \mapsto \mathbb{Z}$ be Δ_f-near concave and $g\colon [m] \mapsto \mathbb{Z}$ be Δ_g-near concave. Then, $h := \textsc{MaxConv}(f, g)$ is Δ_h-near concave with $\Delta_h \leq \max\{\Delta_f, \Delta_g\}$.*

Proof. Let $\breve{f}, \breve{g}$ be pointwise minimal concave functions with $\breve{f} \geq f$, $\breve{g} \geq g$ and let $\breve{h} := \textsc{MaxConv}(\breve{f}, \breve{g})$. We will show that $\breve{h} \geq h \geq \breve{h} - \max\{\Delta_f, \Delta_g\}$, which implies the statement.

To show that $\breve{h} \geq h$, fix $k \in [n+m]$ and let i^* be a witness for $h(k)$, i.e., $h(k) = f(i^*) + g(k - i^*)$. Then, $\breve{h}(k) \geq \breve{f}(i^*) + \breve{g}(k - i^*) \geq f(i^*) + g(k - i^*) = h(k)$. So $\breve{h} \geq h$.

To show that $h \geq \breve{h} - \Delta$ for $\Delta := \max\{\Delta_f, \Delta_g\}$, fix $k \in [n+m]$ and let i^* be a witness for $\breve{h}(k)$, i.e., $\breve{h}(k) = \breve{f}(i^*) + \breve{g}(k - i^*)$. Note that $\breve{f}$ is piecewise a linear interpolation between points on f. In particular, there exist $i_L \leq i^* \leq i_R$ such that $\breve{f}(i_L) = f(i_L)$, $\breve{f}(i_R) = f(i_R)$ and $\breve{f}(i)$ is linear for $i \in [i_L \mathinner{..} i_R]$. Similarly, for $j^* := k - i^*$ there exist $j_L \leq j^* \leq j_R$ such that $\breve{g}(j_L) = g(j_L)$, $\breve{g}(j_R) = g(j_R)$ and $\breve{g}(j)$ is linear for $j \in [j_L \mathinner{..} j_R]$. We pick the maximum i_L, j_L and minimum i_R, j_R with this property.

Let $\hat{\imath}_L := \max\{i_L, k - j_R\}$, $\hat{\imath}_R := \min\{i_R, k - j_L\}$. Observe that the function $\breve{s}(i) := \breve{f}(i) + \breve{g}(k - i)$ is linear for $i \in [\hat{\imath}_L \mathinner{..} \hat{\imath}_R]$, and that $\hat{\imath}_L \leq i^* \leq \hat{\imath}_R$. Moreover, by definition of $\breve{h}$ we have that $\breve{s}(i) = \breve{f}(i) + \breve{g}(k - i) \leq \breve{h}(k)$ for $i \in [\hat{\imath}_L \mathinner{..} \hat{\imath}_R]$. Since i^* is a witness of $\breve{h}(k)$, we have $\breve{s}(i^*) = \breve{h}(k)$. Combining the above, we obtain that $\breve{s}(i) = \breve{h}(k)$ for all $i \in [\hat{\imath}_L \mathinner{..} \hat{\imath}_R]$. In particular, $\breve{f}(\hat{\imath}_L) + \breve{g}(k - \hat{\imath}_L) = \breve{h}(k)$, and thus

$$\begin{aligned} h(k) &\geq f(\hat{\imath}_L) + g(k - \hat{\imath}_L) \\ &= \breve{f}(\hat{\imath}_L) + \breve{g}(k - \hat{\imath}_L) + (f(\hat{\imath}_L) - \breve{f}(\hat{\imath}_L)) + (g(k - \hat{\imath}_L) - \breve{g}(k - \hat{\imath}_L)) \\ &= \breve{h}(k) + (f(\hat{\imath}_L) - \breve{f}(\hat{\imath}_L)) + (g(k - \hat{\imath}_L) - \breve{g}(k - \hat{\imath}_L)). \end{aligned} \tag{2.10}$$

Finally, since $\hat{\imath}_L \in \{i_L, k - j_R\}$ and $f(i_L) = \breve{f}(i_L)$ and $g(j_R) = \breve{g}(j_R)$, one of the two last summands in (2.10) must be 0. Using the near-concavity of f and g, we can bound the other summand by $f(\hat{\imath}_L) - \breve{f}(\hat{\imath}_L) \geq -\Delta_f$ or $g(k - \hat{\imath}_L) - \breve{g}(k - \hat{\imath}_L) \geq -\Delta_g$. This yields $h(k) \geq \breve{h}(k) - \Delta_f$ or $h(k) \geq \breve{h}(k) - \Delta_g$. In any case, we conclude that $h(k) \geq \breve{h}(k) - \max\{\Delta_f, \Delta_g\}$ holds for every $k \in [n+m]$. □

The next lemma shows that the sequences we combine in Line 13 are near-concave.

Lemma 2.19 (Near Concavity)**.** *For every level $\ell \in [1 \mathinner{..} q]$ and every $j \in [1 \mathinner{..} q/2^\ell]$, it holds that $C_j^\ell[J^\ell]$ is $p_{\max}$-near concave.*

Proof. We prove the statement using induction. Focus in the base case $\ell = 0$. For each $j \in [1 \mathinner{..} q]$, we have that $C_j^0[J^0] = \mathcal{P}_{\mathcal{I}_j^0}[J^0]$. In what follows, we argue that $\mathcal{P}_{\mathcal{I}_j^0}$ is $p_{\max}$-near concave. Consider the fractional greedy solution for Knapsack: sort the items $(p_1, w_1), \dots, (p_m, w_m)$ in $\mathcal{I}_j^0$ non-decreasingly by their profit-to-weight ratio, i.e., so that $p_1/w_1 \geq p_2/w_2 \geq \cdots \geq p_m/w_m$. Let $M := \sum_{i=1}^m w_i$. Then, construct the sequence $\tilde{\mathcal{P}}[0\,..\,M]$ by setting breakpoints

$$\tilde{\mathcal{P}}[0] = 0,\ \tilde{\mathcal{P}}[w_1] = p_1,\ \tilde{\mathcal{P}}[w_1 + w_2] = p_1 + p_2,\ \ldots,\ \tilde{\mathcal{P}}[w_1 + \cdots + w_m] = p_1 + \cdots + p_m,$$

and a linear interpolation between every pair of consecutive breakpoints. In this way, $\tilde{\mathcal{P}}[i]$ corresponds the optimal solution to the *fractional* version of Knapsack with capacity i, i.e., in the setting where items can be fractionally packed in a solution.

$\triangleright$ Claim 2.20. The sequence $\tilde{\mathcal{P}}$ is concave, and it holds that $\tilde{\mathcal{P}}[i] \geq \mathcal{P}_{\mathcal{I}_j^0}[i] \geq \tilde{\mathcal{P}}[i] - p_{\max}$ for every $i \in [M]$.

Proof. For each $i \in [1\,..\,M-1]$ it holds that $\tilde{\mathcal{P}}[i] - \tilde{\mathcal{P}}[i-1] \geq \tilde{\mathcal{P}}[i+1] - \tilde{\mathcal{P}}[i]$ since the slopes of the linear pieces between breakpoints are non-decreasing due to the sorting by profit-to-weight ratio. This means that $\tilde{\mathcal{P}}$ is concave.

For each $i \in [M]$, it holds that $\tilde{\mathcal{P}}[i] \geq \mathcal{P}_{\mathcal{I}_j^0}[i]$ since $\tilde{\mathcal{P}}[i]$ is the optimal solution of the fractional Knapsack. Moreover, observe that the solution attaining $\tilde{\mathcal{P}}[i]$ contains at most one item allocated fractionally. By removing that item, we obtain a feasible (integral) solution to the Knapsack of capacity i, and the profit is reduced by at most $p_{\max}$. This implies that $\mathcal{P}_{\mathcal{I}_j^0}[i] \geq \tilde{\mathcal{P}}[i] - p_{\max}$. $\triangleleft$

By Claim 2.20, we conclude that $\mathcal{P}_{\mathcal{I}_j^0}[0\,..\,M]$ is $p_{\max}$-near concave (see Definition 2.1), and therefore $C_j^0[J^0]$ is as well, which completes the proof of the base case.

For the inductive step, consider a level $\ell > 0$. Fix a $j \in [1\,..\,q/2^\ell]$. By the inductive hypothesis, $C_{2j-1}^{\ell-1}[J^{\ell-1}]$ and $C_{2j}^{\ell-1}[J^{\ell-1}]$ are $p_{\max}$-near concave. Thus, by Lemma 2.18 we obtain that $C_j^\ell[J^\ell] = \textsc{MaxConv}(C_{2j-1}^{\ell-1}[J^{\ell-1}], C_{2j}^{\ell-1}[J^{\ell-1}])$ is $p_{\max}$-near concave, completing the proof. $\square$

Lemma 2.21. *Fix a level $\ell \in [1\,..\,q]$ and an iteration $j \in [1\,..\,q/2^\ell]$. The computation of C_j^ℓ in Line 13 takes time $\widetilde{O}(p_{\max}\sqrt{\Delta 2^\ell})$*

Proof. By Lemma 2.19, the sequences $C_{2j-1}^{\ell-1}[J^{\ell-1}], C_{2j}^{\ell-1}[J^{\ell-1}]$ are $p_{\max}$-near concave. Thus, using Corollary 2.17, we can compute their max-plus convolution in time

$$\widetilde{O}(p_{\max}|J^\ell|) = \widetilde{O}(p_{\max}\sqrt{\Delta 2^\ell}),$$

where we used $\eta = \widetilde{O}(1)$. $\square$

Lemma 2.22 (Running Time of Algorithm 3)**.** *Algorithm 3 runs in time*

$$\widetilde{O}((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max}).$$

Proof. Recall that $q = \min\{\,(n/p_{\max})^{2/3}(W/w_{\max})^{1/3}, W/w_{\max}\,\}$ (up to a factor of 2). Since $W \leq nw_{\max}$, we have that $q \leq n$. Moreover, since we assume without loss of generality that $w_{\max} \leq n$, note that $q < 1$ if and only if $q = (n/p_{\max})^{2/3}(W/w_{\max})^{1/3} < 1$.

This implies that $p_{\max} > n\sqrt{W/w_{\max}}$. But in this case, the claimed running time is $\Omega(nW)$, so the standard $O(nW)$ dynamic programming algorithm (Fact 2.12) already achieves our time bound. Thus, we can assume without loss of generality that $1 \le q \le n$, i.e., q is a valid choice for the number of groups in which we split the item set $\mathcal{I}$.

We start bounding the running time of the base case, i.e., the computation of the arrays C_j^0 for $j \in [1\,..\,q]$ in Line 5. By Fact 2.12, and the definition $\Delta = w_{\max}W/q$ this takes time

$$O\left(\sum_{j=1}^{q} |\mathcal{I}_j^0|(\tfrac{W}{q} + \sqrt{\Delta}\eta)\right) = O\left(n(\tfrac{W}{q} + \sqrt{\Delta}\eta)\right) = \widetilde{O}\left(n\frac{W}{q} + n\sqrt{\frac{w_{\max}W}{q}}\right). \tag{2.11}$$

Now we bound the time of the combination step done in Lines 9 to 14. At level $\ell \in [1\,..\,q]$ and iteration $j \in [1\,..\,q/2^\ell]$ the execution of Line 13 takes time $\widetilde{O}(p_{\max}\sqrt{\Delta 2^\ell})$ by Lemma 2.21. Thus, we can bound the overall time as

$$\sum_{\ell=1}^{\log q}\sum_{j=1}^{q/2^\ell} \widetilde{O}(p_{\max}\sqrt{\Delta 2^\ell}) = \sum_{\ell=1}^{\log q} \frac{q}{2^\ell}\widetilde{O}\left(p_{\max}\sqrt{\frac{w_{\max}W}{q}2^\ell}\right) = \sum_{\ell=1}^{\log q} \widetilde{O}\left(p_{\max}\sqrt{\frac{q w_{\max}W}{2^\ell}}\right),$$

since this is a geometric series, it is bounded by the first term $\widetilde{O}(p_{\max}\sqrt{q w_{\max}W})$. Combining this with (2.11), we obtain overall time

$$\widetilde{O}\left(p_{\max}\sqrt{q w_{\max}W} + n\frac{W}{q} + n\sqrt{\frac{w_{\max}W}{q}}\right).$$

Recalling that $q = \Theta(\min\{\, (n/p_{\max})^{2/3}(W/w_{\max})^{1/3}, W/w_{\max}\,\})$, we obtain overall time

$$\widetilde{O}((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max} + (p_{\max}W)^{1/3}(nw_{\max})^{2/3}).$$

Finally, note that using the inequality $\sqrt{xy} \le x + y$ for all $x, y \ge 0$, we have

$$\begin{aligned}(p_{\max}W)^{1/3}(nw_{\max})^{2/3} &= \sqrt{(p_{\max}W)^{2/3}(nw_{\max})^{1/3} nw_{\max}} \\ &\le O((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max}).\end{aligned}$$

Thus, the overall running time is $\widetilde{O}((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max})$, as claimed. □

Proof of Main Theorem 1.2. Run Algorithm 3. Lemma 2.15 guarantees that $\mathcal{I}_1^{\log q}[W] = \mathrm{OPT}$ with probability at least $1 - 1/n^2$, which proves correctness. The running time is immediate from Lemma 2.22. Observe that we can obtain success probability $1 - 1/n^c$ for any constant $c \ge 2$ by repeating the algorithm $c/2$ times. □

Proof Sketch of Main Theorem 1.3 Our presentation focused on proving Main Theorem 1.2. The proof of the symmetric variant stated in Main Theorem 1.3 is very similar, thus we only sketch the required changes. Essentially, we need to exchange profits with weights everywhere, which in turn means exchanging max-plus convolutions by min-plus convolutions. In more detail: Instead of working with the profit sequence $\mathcal{P}_{\mathcal{I}}$, we work with the weight sequence $\mathcal{W}_{\mathcal{I}}$, where the entry $\mathcal{W}_{\mathcal{I}}[j]$ stores the minimum weight of a solution with profit at least j. We do not know OPT, but we can compute an approximation $\tilde{V}$ satisfying $\tilde{V} - p_{\max} \leq \mathrm{OPT} \leq \tilde{V}$ in linear time (see e.g. [KPP04, Theorem 2.5.4]). In the algorithm, we exchange all occurrences of $w_{\max}$ by $p_{\max}$ and all occurrences of W by $\tilde{V}$. With these changes, the functions C_j^ℓ are now $w_{\max}$-near convex (instead of $p_{\max}$-near concave) so we use Main Theorem 1.7 directly instead of Corollary 2.17. In this way, we obtain the array $C_1^{\log q}[\tilde{V} - p_{\max} \,..\, \tilde{V}] = \mathcal{W}_{\mathcal{I}}[\tilde{V} - p_{\max} \,..\, \tilde{V}]$. Then, we can infer OPT as the largest $i \in [\tilde{V} - p_{\max} \,..\, \tilde{V}]$ such that $\mathcal{W}_{\mathcal{I}}[i] \leq W$.

Reconstructing an optimal solution So far we were only concerned with returning the optimal profit of a given Knapsack instance. To reconstruct a solution $x \in \{0, 1\}^n$ such that $p_{\mathcal{I}}(x) = \mathrm{OPT}$, we proceed as follows. After running Algorithm 3, we obtain the sequences $C_1^\ell[J^\ell]$ for every $\ell \in [\log q]$ and $j \in [1 \,..\, q/2^\ell]$. For the output entry $C_1^{\log q}[W]$, we find a witness $i \in J^{\log q - 1}$, i.e., an index i such that $C_1^{\log q-1}[i] + C_2^{\log q-1}[W - i] = C_1^{\log q}[W]$. This can be done in time $|J^{\log q-1}| = \widetilde{O}(\sqrt{\Delta q/2})$ by simply trying all possibilities. Then, we continue recursively finding witnesses for i and $W - i$. Eventually, we reach one entry in each array C_j^0 for $j \in [1 \,..\, q]$. Note that this takes time proportional to the length of all sequences $\sum_{\ell=0}^{\log q} q/2^\ell \cdot O(|J^\ell|) = \widetilde{O}(q\sqrt{\Delta}) = \widetilde{O}(\sqrt{q w_{\max} W}) \leq \widetilde{O}(n w_{\max})$, where the last step uses $q \leq W/w_{\max}$ and $W \leq n w_{\max}$. Finally, observe each array C_j^0 was computed using the standard dynamic programming algorithm of Fact 2.12, which allows to retrieve a solution for a fixed entry $C_j^0[i]$ in the same time it takes to compute it. Thus, we can retrieve the optimal solution with no extra overhead on the overall running time.

3 Algorithms via Bounded Monotone Min-Plus Convolution

This chapter contains our algorithmic results obtained via Bounded Monotone min-plus convolution. The content is based on our publication [BC22].

Organization The outline of the chapter is as follows. We start with preliminaries in Section 3.1. In Section 3.2 we give our exact algorithm for Unbounded Knapsack, proving Main Theorem 1.4. In Section 3.3 we give our exact algorithm for Knapsack, establishing Main Theorem 1.1. Then, we turn to approximation schemes for Unbounded Knapsack in Section 3.4, where we prove Theorem 1.5 and Main Theorem 1.6. Finally, in Section 3.5 we give an equivalence (see Theorem 3.32) between Bounded Monotone min-plus convolution and the problems we studied.

3.1 Preliminaries

We use the notation $\widetilde{O}(T) = \bigcup_{c \geq 0} O(T \log^c(T))$ to suppress polylogarithmic factors.

Let $\mathcal{I}$ be a set of n items with weights $w_1, \ldots, w_n \in \mathbb{N}$ and profits $p_1, \ldots, p_n \in \mathbb{N}$. We identify the item set $\mathcal{I}$ with the set $\{1, \ldots, n\}$. We denote by $x \in \mathbb{N}^n$ a multiset of items, where x_i is the number of copies of the i-th item[1]. Sometimes we refer to x as a *solution*. Furthermore, we write $p_{\mathcal{I}}(x)$ for the total profit of x, i.e., $p_{\mathcal{I}}(x) := \sum_i x_i \cdot p_i$. Similarly, we write $w_{\mathcal{I}}(x) := \sum_i x_i \cdot w_i$ for the weight of x. When the item set $\mathcal{I}$ is clear from context, we drop the subscript and simply write $p(x)$ and $w(x)$. We denote the number of items contained in a solution x by $\|x\|_1 := \sum_i x_i$. A solution x is *feasible* if it satisfies the constraint $w(x) \leq W$. We denote by $p_{\max} := \max_i p_i$ the maximum profit and by $w_{\max} := \max_i w_i$ the maximum weight of the input set $\mathcal{I}$.

Bounded Monotone MaxPlus Convolution Recall that the min-plus convolution of two integer sequences $A[0\,..\,n], B[0\,..\,n]$ is the sequence $C[0\,..\,2n]$ where $C[k] = \min_{i+j=k} A[i] + B[k]$. For the special case when A and B are monotone increasing and have bounded entries, Chi, Duan, Xie and Zhang proved the following remarkable result:

1. When we work with Knapsack, we restrict $x \in \{0, 1\}^n$

Theorem 3.1 (Bounded Monotone MinPlus Conv [Chi+22])**.** *There is an algorithm that given monotone increasing sequences $A[0\,..\,n]$ and $B[0\,..\,n]$ with entries $A[i], B[i] \in [O(n)]$ for all $i \in [n]$, computes $\textsc{MinConv}(A, B)$ in expected time $\widetilde{O}(n^{1.5})$.*

For our applications, we shall be concerned with the max-plus convolution of bounded sequences, formally defined as follows.

Problem 3.2 (Bounded Monotone MaxPlus Conv)**.** *Given monotone non-decreasing sequences $A[0\,..\,n]$ and $B[0\,..\,n]$ with entries $A[i], B[i] \in [O(n)] \cup \{-\infty\}$ for all $i \in [n]$. The task is to compute their max-plus convolution $C = \textsc{MaxConv}(A, B)$.*

The following simple corollary shows that the result of Chi, Duan, Xie and Zhang can be used to solve Bounded Monotone MaxPlus Conv:

Corollary 3.3 (Bounded Monotone MaxPlus Conv [Chi+22])**.** *There is an algorithm that solves Bounded Monotone MaxPlus Conv in expected time $\widetilde{O}(n^{1.5})$.*

Proof. We describe how to reduce Bounded Monotone MaxPlus Conv to min-plus convolution on monotone increasing sequences and values in $[O(n)]$ via a simple chain of reductions:

- *Removing $-\infty$:* Let $A[0\,..\,n], B[0\,..\,n]$ be an instance of Bounded Monotone MaxPlus Conv, i.e., A, B are monotone non-decreasing and $A[i], B[i] \in [O(n)] \cup \{-\infty\}$. We start by reducing it to an equivalent instance of max-plus convolution on monotone non-decreasing sequences and values in $[O(n)]$ (i.e. we remove the $-\infty$ entries). Let Δ be the maximum entry of A and B. Construct a new sequence $A'[0\,..\,n]$ where $A'[i] := 0$ if $A[i] = -\infty$, and $A'[i] := A[i] + 2\Delta$ otherwise. Construct $B'[0\,..\,n]$ from B in the same way. Note that A' and B' are monotone non-decreasing and have values in $[O(n)]$. Let $C := \textsc{MaxConv}(A, B)$ and $C' := \textsc{MaxConv}(A', B')$. Observe that we can infer the values of any entry $C[k]$ from C': if $C'[k] \leq 3\Delta$ then $C[k] = -\infty$ and otherwise $C[k] = C'[k] - 4\Delta$.

- *Reducing to max-plus convolution on non-increasing sequences:* Now we reduce an instance $A[0\,..\,n]$, $B[0\,..\,n]$ of max-plus convolution on monotone non-decreasing sequences and values in $[O(n)]$ to an instance of min-plus convolution on monotone non-increasing sequences and values in $[O(n)]$. Let Δ be the maximum entry of A and B. Construct two new sequences A' and B' by setting $A'[i] := \Delta - A[i]$ and $B'[i] := \Delta - B[i]$. Then A' and B' are monotone non-increasing and given their min-plus convolution we can easily infer the max-plus convolution of A and B.

- *Reducing to min-plus convolution on increasing sequences:* Next, we reduce an instance $A[0\,..\,n]$, $B[0\,..\,n]$ of min-plus convolution on monotone non-increasing sequences and values in $[O(n)]$ to an instance of min-plus convolution on increasing sequences and values in $[O(n)]$. Construct two new sequences A' and B' by reversing and adding a linear function to A and B, i.e., set $A'[i] := A[n - i] + i$ and $B'[i] := B[n - i] + i$ for every $i \in [n]$. Note that A' and B' are monotone increasing sequences, and given their min-plus convolution we can infer the min-plus convolution of A and B.

Combining the reductions above, we conclude that Bounded Monotone MaxPlus Conv can be reduced in linear time to min-plus convolution on monotone increasing sequences with values in $[O(n)]$. Applying Theorem 3.1 yields the corollary. $\square$

Witnesses Let $A[0\,..\,n], B[0\,..\,n]$ and let $C := \textsc{MaxConv}(A, B)$. Given $k \in [2n]$, we say that $i \in [n]$ is a *witness* for $C[k]$ if $C[k] = A[i] + B[k-i]$. We say that $M[0\,..\,2n]$ is a *witness array*, if each entry $M[k]$ contains some witness for $C[k]$.

For the general case of max-plus convolution (i.e. not the bounded monotone version) it is well known (e.g. [Sei95; Alo+92]) that computing the witness array has the same time complexity as max-plus convolution, up to a polylog(n) overhead. This reduction does not immediately apply to Bounded Monotone MaxPlus Conv because the sequences might not remain monotone. However, in what follows we show how to make it work.

Fix an instance $A[0\,..\,n], B[0\,..\,n]$ of Bounded Monotone MaxPlus Conv and let $C := \textsc{MaxConv}(A, B)$. To compute the witness array $M[0\,..\,2n]$, we first show that if an entry $C[k]$ has a unique witness then we can easily find it. Then we reduce to the unique witness case with randomization.

Lemma 3.4. *If Bounded Monotone MaxPlus Conv on length-n sequences can be computed in time $T(n)$, then in time $\widetilde{O}(T(n))$ we can compute an array $U[0\,..\,2n]$ such that for every $k \in [2n]$, if $C[k]$ has a unique witness $M[k]$, then $U[k] = M[k]$. The output $U[k]$ is undefined otherwise.*

Proof. We will describe how to compute the unique witness bit by bit. For each bit position $b \in [\lceil \log n \rceil]$ let $i_b \in \{0, 1\}$ be the b-th bit of i. Construct sequences A_b, B_b defined as $A_b[i] := 2A[i] + i + i_b$ and $B_b[i] := 2B[i] + i$ for $i \in [n]$. Note that A_b, B_b are still monotone non-decreasing and have entries bounded by $O(n)$. Compute $C_b := \textsc{MaxConv}(A_b, B_b)$. For each $k \in [2n]$, we set the b-th bit of $U[k]$ to $C_b[k] \bmod 2$. It is not hard to see that for those entries $C[k]$ which have unique witnesses $U[k]$, this procedure indeed gives the b-th bit of $U[k]$. Indeed, note that because we double every entry in A_b, B_b and add a linear function, adding i_b does not change the maximizer. Therefore, if $C[k]$ has a unique witness then $C'[k]$ has a unique witness whose b-th bit can be read from the least significant bit of $C'[k]$. Thus, by repeating this over all bit positions $b \in [\lceil \log n \rceil]$, we compute the entire array of unique witnesses U using $O(\log n)$ invocations to Bounded Monotone MaxPlus Conv, as desired. $\square$

Lemma 3.5 (Witness Finding). *If Bounded Monotone MaxPlus Conv can be computed in time $T(n)$, then a witness array $M[0\,..\,2n]$ can be computed in time $\widetilde{O}(T(n))$.*

Proof. Fix a set $S \subseteq [n]$. We say that an entry $C[k]$ gets *isolated* by S if the number of witnesses of $C[k]$ in S is exactly one. We will now describe how to find the witnesses of all entries isolated by S (the idea and argument is similar as in the proof of Lemma 3.4). Construct sequences A', B' where for each $i \in [n]$ we set

$$A'[i] := \begin{cases} 2A[i] + i + 1 & \text{if } i \in S \\ 2A[i] + i & \text{otherwise} \end{cases}$$

and $B'[i] := 2B[i] + i$. Note that these sequences are monotone non-decreasing and have entries bounded by $O(n)$. Let $C' := \textsc{MaxConv}(A', B')$. We claim that if an entry $C[k]$ is isolated by S, then $C'[k]$ has a unique witness. To see this, note that if no witness of $C[k]$ gets included in S, then we have that $C'[k] = 2C[k] + k$. If at least one witness gets included in S, then $C'[k] = 2C[k] + k + 1$. In particular, if a witness gets isolated then $C'[k]$ will have a unique witness, as claimed. Thus, by Lemma 3.4 we can compute in time $\widetilde{O}(T(n))$ an array $U[0\,..\,2n]$ which contains the witnesses of all entries that are isolated by S. Note that some entries of U might be undefined, but we can simply check in time $O(n)$ which entries of U are true witnesses, by iterating over $k \in [2n]$ and checking whether the equality $C[k] = A[U[k]] + B[k - U[k]]$ holds.

Now we show how to select appropriate sets S. Fix an entry $C[k]$ and denote by $R \in [n]$ its number of witnesses. We sample $S \subseteq [n]$ by including each element $i \in [n]$ independently with probability $p := 2^{-\alpha}$ where $\alpha \in \mathbb{N}$ is chosen such that $2^{\alpha-2} \leq R \leq 2^{\alpha-1}$. Let X be the random variable counting the number of witnesses of $C[k]$ that get sampled in S. By keeping the first two terms in the inclusion-exclusion formula we have that $\mathbb{P}[X \geq 1] \geq p \cdot R - \binom{R}{2} p^2$, and by a union bound $\mathbb{P}[X \geq 2] \leq \binom{R}{2} p^2$. Thus,

$$\mathbb{P}[X = 1] = \mathbb{P}[X \geq 1] - \mathbb{P}[X \geq 2] \geq p \cdot R(1 - p \cdot R) \geq 1/8$$

where the last inequality holds because $1/8 \leq p \cdot R \leq 1/4$ due to the choice of α. In particular, S isolates $C[k]$ with probability at least $1/8$.

We now put the pieces together. Iterate over the $O(\log n)$ possible values for α. Sample a set S and find all witnesses of entries isolated by S as described earlier in time $\widetilde{O}(T(n))$. As we argued above, if $C[k]$ has R witnesses and $2^{\alpha-2} \leq R \leq 2^{\alpha-1}$, then $C[k]$ gets isolated with constant probability. Thus, by repeating this step with the same α for $O(\log n)$ freshly sampled sets S we find a witness for all such entries $C[k]$ with probability at least $1 - 1/\mathrm{poly}(n)$. Combining the results across iterations we obtain the array of witnesses $M[0\,..\,2n]$ in time $\widetilde{O}(T(n))$, as desired.

Finally, we note that this procedure can be derandomized with a polylogarithmic overhead using ε-biased sets, as in [AN96]. □

Niceness assumptions on time bounds In order to phrase our reductions, we make the following assumptions about time bounds.

Assumption 3.6 (Niceness Assumptions)**.** *For all time bounds $T(n)$ in this chapter, we assume that*

1. $T(\widetilde{O}(n)) \leq \widetilde{O}(T(n))$,
2. $k \cdot T(n) \leq O(T(kn))$ *for any* $k, n \geq 1$.

Note that these assumptions are satisfied by all *natural* time bounds of polynomial-time or pseudopolynomial-time algorithms. In particular, it holds for all functions of the form $T(n) = \Theta(n^\alpha \log^\beta n)$ for any constants $\alpha \geq 1, \beta \geq 0$.

3.2 Exact algorithm for Unbounded Knapsack

This section is devoted to prove the following theorem:

Theorem 3.7. *If Bounded Monotone MaxPlus Conv on length-n sequences can be solved in time $T(n)$, then Unbounded Knapsack can be solved in time $\widetilde{O}(n + T(p_{\max} + w_{\max}))$, where $p_{\max}$ is the largest profit of any item and $w_{\max}$ is the largest weight of any item.*

Note that Main Theorem 1.4 follows as an immediate corollary of Theorem 3.7 by plugging in Chi, Duan, Xie and Zhang's algorithm (Corollary 3.3).

For the rest of this section, fix an instance $(\mathcal{I}, W)$ of Unbounded Knapsack. We define the profit sequence $\mathcal{P}_{\mathcal{I}}[\cdot]$, where for each $j \in \mathbb{N}$, the entry $\mathcal{P}_{\mathcal{I}}[j]$ is defined the maximum profit achievable with capacity j, i.e.,

$$\mathcal{P}_{\mathcal{I}}[j] := \max\{\, p_{\mathcal{I}}(x) \mid x \in \mathbb{N}^n, w_{\mathcal{I}}(x) \leq j \,\}.$$

Note that $\mathcal{P}_{\mathcal{I}}[0] = 0$.

Given an integer $k \geq 0$ we also define the sequence $\mathcal{P}_{\mathcal{I},k}[0\,..\,W]$, where we restrict to solutions with at most 2^k items, for any nonnegative integer k. That is, for any $j \in [W]$ we set

$$\mathcal{P}_{\mathcal{I},k}[j] := \max\{\, p_{\mathcal{I}}(x) \mid x \in \mathbb{N}^n, w(x) \leq j, \|x\|_1 \leq 2^k \,\}.$$

When $\mathcal{I}$ is clear from context, we will drop the subscript and write $\mathcal{P}[0\,..\,W]$ and $\mathcal{P}_k[0\,..\,W]$. Throughout this section we set $\Delta := p_{\max} + w_{\max}$.

We start with a high level overview of the proof of Theorem 3.7.

Proof Overview Since each item has weight at least 1, any feasible solution consists of at most W items. Thus, our goal is to compute the value $\mathcal{P}_{\lceil \log W \rceil}[W] = \mathrm{OPT}$. The natural approach is to use dynamic programming: since $\mathcal{P}_0$ consists of solutions of at most one item, it can be computed in time $O(n)$. For $i > 0$ we can compute $\mathcal{P}_i[0\,..\,W]$ by taking the max-plus convolution of $\mathcal{P}_{i-1}[0\,..\,W]$ with itself. This gives an algorithm in time $O(W^2 \log W)$.

Jansen and Rohwedder [JR23] and Axiotis and Tzamos [AT19] showed that instead of convolving sequences of length W, it suffices to convolve only $O(w_{\max})$ entries of $\mathcal{P}_{i-1}$ in each iteration. This improves the running time to $O(w_{\max}^2 \log W)$ by using the naive algorithm for max-plus convolution. In more detail, the approach of Jansen and Rohwedder [JR23] is the following. Suppose x is the optimal solution for a target value $\mathcal{P}_i[j]$. They showed that x can be split into two solutions x_1, x_2 such that (i) the number of items in each part is at most 2^{i-1} and (ii) the difference between the weights of both parts is at most $O(w_{\max})$. Thus, (i) guarantees that both x_1 and x_2 are optimal solutions for two entries of $\mathcal{P}_{i-1}$, and (ii) implies that these entries lie in an interval in $\mathcal{P}_{i-1}$ of length $O(w_{\max})$. In this way, they can afford to perform the max-plus convolution of only $O(w_{\max})$ entries in $\mathcal{P}_{i-1}$.

To show the existence of such a partitioning of x they used Steinitz' Lemma [Ste13]. This lemma states that any collection of m vectors in $\mathbb{R}^d$ with infinity norm at most 1, whose sum is 0, can be permuted such that every prefix sum has norm at most $O(d)$ (see Lemma 3.8 for the precise statement). The partitioning of x follows from Steinitz' Lemma by taking the weights of the items picked by x as 1-dimensional vectors. The usage of Steinitz' Lemma to reduce the number of states in dynamic programs was pioneered by Eisenbrand and Weismantel [EW20], and further refined by Jansen and Rohwedder [JR23].

In our algorithm, we use Steinitz' Lemma in a similar way to split the number of items and the weight of x, but additionally we use it to ensure that the profits of the solutions x_1, x_2 differ by at most $O(p_{\max})$ (see Lemma 3.9). In this way, by carefully handling the subproblems $\mathcal{P}_{i-1}$ we can enforce that the values of the $O(w_{\max})$ entries that need to be convolved have values in a range of size $O(p_{\max})$. Since the arrays $\mathcal{P}_i$ are monotone non-decreasing, we can then apply the algorithm for Bounded Monotone MaxPlus Conv, and thus handle each subproblem in time $\widetilde{O}((p_{\max} + w_{\max})^{1.5})$. The resulting total time to compute $\mathcal{P}_i[W]$ is $O(n + k \cdot T(\Delta))$. With additional preprocessing we ensure that $k = O(\log \Delta)$ (see Lemma 3.13), turning the running time into $\widetilde{O}(n + T(\Delta))$.

3.2.1 Preparations

We need to show that when computing the optimal answer for some entry $\mathcal{P}_i[j]$, we can split it in such a way that both its total profit and its total weight are roughly halved. Our main tool to show this is the Steinitz Lemma [GS80; Ste13]. A beautiful proof for it can be found in [Mat10, Miniature 20].

Lemma 3.8 ([Ste13; GS80, Steinitz Lemma])**.** *Let $\|\cdot\|$ be a norm in $\mathbb{R}^m$ and let M be an arbitrary collection of t vectors in $\mathbb{R}^m$ such that $\|v\| \leq 1$ for every $v \in M$ and $\sum_{v \in M} v = 0$. Then, it is possible to permute the vectors in M into a sequence $(v_1, \dots, v_t)$ such that $\|v_1 + \cdots + v_k\| \leq m$ holds for every $k \in [t]$.*

We use the Steinitz Lemma to argue that the items in a solution can be split in two parts in such a way that both the total profit and the total weight are roughly halved:

Lemma 3.9 (Splitting Lemma)**.** *Let $i \geq 1$ and consider a solution $x \in \mathbb{N}^n$ with $\|x\|_1 \leq 2^i$. Then there is a partition of x into two solutions $x_1, x_2 \in \mathbb{N}^n$ with the following properties:*

1. *(Splitting of Items) $\|x_1\|_1, \|x_2\|_1 \leq 2^{i-1}$ and $x = x_1 + x_2$,*
2. *(Approximate Splitting of Weight) $|w(x_1) - \frac{1}{2}w(x)| \leq 2\Delta$ and $|w(x_2) - \frac{1}{2}w(x)| \leq 2\Delta$,*
3. *(Approximate Splitting of Value) $|p(x_1) - \frac{1}{2}p(x)| \leq 2\Delta$ and $|p(x_2) - \frac{1}{2}p(x)| \leq 2\Delta$.*

Proof. Let $t := \|x\|_1 \leq 2^i$. First assume that t is even; we will remove this assumption later. Write $x = \sum_{j=1}^{t} x^{(j)}$ where each $x^{(j)}$ corresponds to one copy of some item, i.e. $\|x^{(j)}\|_1 = 1$, and set $v^{(j)} = \binom{w(x^{(j)})}{p(x^{(j)})}$. Note that $\|v^{(j)}\|_\infty \leq \Delta$. By applying the Steinitz

Lemma on the vectors $v^{(j)} - \frac{1}{t}\binom{w(x)}{p(x)}$ (after normalizing by Δ), we can assume that the $v^{(j)}$'s are ordered such that

$$\left\| \sum_{j=1}^{t/2} v^{(j)} - \frac{1}{2}\begin{pmatrix} w(x) \\ p(x) \end{pmatrix} \right\|_\infty \leq 2\Delta. \tag{3.1}$$

Fix this ordering, and let $x_1 = x^{(1)} + \ldots + x^{(t/2)}$, corresponding to $v^{(1)}, \ldots, v^{(t/2)}$, and let $x_2 = x^{(t/2+1)} + \ldots + x^{(t)}$, corresponding to the remaining vectors $v^{(t/2+1)}, \ldots, v^{(t)}$. We now check that x_1, x_2 satisfy the properties of the lemma:

- Property 1 is clearly satisfied by construction.
- For property 2, note that (3.1) implies $|w(x_1) - \frac{1}{2}w(x)| \leq 2\Delta$. Since $w(x_2) = w(x) - w(x_1)$, we have that $|w(x_2) - \frac{1}{2}w(x)| = |\frac{1}{2}w(x) - w(x_1)| \leq 2\Delta$.
- Property 3 follows in the same way as property 2.

If t is odd, then $t + 1 \leq 2^i$, so we can add a dummy vector $x^{(t+1)} = 0$ with corresponding $v^{(t+1)} := \binom{0}{0}$ and repeat the same argument with $t := t + 1$. □

When we apply Lemma 3.9 to an optimal solution corresponding to an entry of the array $\mathcal{P}_i$, we obtain the following lemma.

Lemma 3.10. *Let $\beta > 0$. For any index $j \in [\beta - 8\Delta \,..\, \beta + 8\Delta] \cap [W]$ there are indices $j_1, j_2 \in [\frac{\beta}{2} - 8\Delta \,..\, \frac{\beta}{2} + 8\Delta] \cap [W]$ with the following properties:*

(i) $j_1 + j_2 = j$,

(ii) $\mathcal{P}_i[j] = \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2]$,

(iii) $|\mathcal{P}_{i-1}[j_1] - \frac{1}{2}\mathcal{P}_i[j]| \leq 2\Delta$ *and* $|\mathcal{P}_{i-1}[j_2] - \frac{1}{2}\mathcal{P}_i[j]| \leq 2\Delta$.

Proof. Let $x \in \mathbb{N}^n$ be an optimal solution for $\mathcal{P}_i[j]$, that is, we have $p(x) = \mathcal{P}_i[j]$, $w(x) \leq j$, and $\|x\|_1 \leq 2^i$. We apply Lemma 3.9 to x and obtain $x_1, x_2 \in \mathbb{N}^n$ such that $x_1 + x_2 = x$. We do a case distinction based on $w(x_1), w(x_2)$:

- $w(x_1), w(x_2) \in [\frac{j}{2} - 4\Delta \,..\, \frac{j}{2} + 4\Delta]$: Let $j_1 := w(x_1)$ and $j_2 := j - j_1$; note that $j_1, j_2 \in [\frac{j}{2} - 4\Delta \,..\, \frac{j}{2} + 4\Delta] \subseteq [\frac{\beta}{2} - 8\Delta \,..\, \frac{\beta}{2} + 8\Delta]$. We argue that $p(x_1) = \mathcal{P}_{i-1}[j_1]$ and $p(x_2) = \mathcal{P}_{i-1}[j_2]$. Indeed, since $w(x_1) = j_1$ the solution x_1 is feasible for weight j_1, so $p(x_1) \leq \mathcal{P}_{i-1}[j_1]$. Similarly, since $w(x_2) = w(x) - w(x_1) \leq j - w(x_1) = j_2$ the solution x_2 is feasible for weight j_2, so $p(x_2) \leq \mathcal{P}_{i-1}[j_2]$. Moreover, by optimality of x we have $p(x_1) + p(x_2) = p(x) = \mathcal{P}_i[j] \geq \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2]$, so we obtain $p(x_1) = \mathcal{P}_{i-1}[j_1]$ and $p(x_2) = \mathcal{P}_{i-1}[j_2]$. Using these equations together with $p(x) = \mathcal{P}_i[j]$, property (ii) follows from $p(x) = p(x_1) + p(x_2)$, property (iii) follows from Property 3 of Lemma 3.9, and property (i) holds by definition of j_2.

- $w(x_1) < \frac{j}{2} - 4\Delta$: Property 2 of Lemma 3.9 implies that $|w(x_1) - w(x_2)| \leq 4\Delta$, and thus $w(x_2) \leq \frac{j}{2}$. Therefore, x_1 and x_2 are feasible for weights $j_1 := \lfloor \frac{j}{2} \rfloor$ and $j_2 := \lceil \frac{j}{2} \rceil$, respectively. Note that $j_1, j_2 \in [\frac{\beta}{2} - (4\Delta + 1) \,..\, \frac{\beta}{2} + (4\Delta + 1)] \subseteq [\frac{\beta}{2} - 8\Delta \,..\, \frac{\beta}{2} + 8\Delta]$. Property (i) is obvious, and properties (ii) and (iii) now follow as in the first case.
- $w(x_1) > \frac{j}{2} + 4\Delta$: Similarly as the previous case, property 2 of Lemma 3.9 implies that $w(x_2) \geq \frac{j}{2}$. Therefore, we have $w(x) = w(x_1) + w(x_2) > j + 4\Delta$, which contradicts the assumption $w(x) \leq j$.
- $w(x_2) < \frac{j}{2} - 4\Delta$ or $w(x_2) > \frac{j}{2} + 4\Delta$: Symmetric to the previous two cases. □

3.2.2 The algorithm

We are now ready to present our algorithm. The idea is to use the Splitting Lemma 3.9 to convolve smaller sequences which are bounded and monotone.

Recall that since any feasible solution contains at most W items, to compute the value of the optimal solution it suffices to compute $\mathcal{P}_k[W]$ where $k := \lceil \log W \rceil$. Our approach is as follows. We do binary search for OPT in the range $[0 \,..\, p_{\max} \cdot W]$. Suppose we have the current guess α. Instead of computing the arrays $\mathcal{P}_i$, we compute *clipped versions* C_i, which has the property that $C_i[j] \geq \alpha$ if and only if $\mathcal{P}_i[j] \geq \alpha$.

We compute C_i as follows: At every step, we only compute the values for $O(\Delta)$ weights $C_i[W \cdot 2^{i-k} - 8\Delta \,..\, W \cdot 2^{i-k} + 8\Delta]$. For the base case $i = 0$, we simply set $C_0[0 \,..\, 8\Delta] := \mathcal{P}_0[0 \,..\, 8\Delta]$. Note that this can be done in time $O(n + \Delta)$ by doing one pass over the item set, since $\mathcal{P}_0$ only considers solutions with at most one item. Moreover, observe that $C_0[0 \,..\, 8\Delta]$ is monotone non-decreasing by definition of $\mathcal{P}_0$.

For the general case $i > 0$ we first compute an array $A_i[W \cdot 2^{i-k} - 8\Delta \,..\, W \cdot 2^{i-k} + 8\Delta]$ by taking the max-plus convolution of $C_{i-1}[W \cdot 2^{i-1-k} - 8\Delta \,..\, W \cdot 2^{i-1-k} + 8\Delta]$ with itself. To obtain $C_i[W \cdot 2^{i-k} - 8\Delta \,..\, W \cdot 2^{i-k} + 8\Delta]$, we clip the values in A_i which are too large, and set to $-\infty$ the values which are too small. This ensures that all values in C_i lie within a range of $O(\Delta)$, except for values that are $-\infty$. Consult Algorithm 4 for the pseudocode.

Lemma 3.11. *Algorithm 4 runs in time $O(n + T(\Delta) \log W)$, where $T(\Delta)$ is the time complexity of Bounded Monotone MaxPlus Conv on sequences of length Δ, and computes a value $C_k[W]$ which satisfies $C_k[W] \geq \alpha$ if and only if* $\mathrm{OPT} = \mathcal{P}_k[W] \geq \alpha$.

Proof. We start with the running time analysis. As mentioned above, the initialization in Line 2 takes time $O(n + \Delta)$ since it suffices to scan the item set. Due to the clipping, at every execution of Line 6 we compute a max-plus convolution of sequences of length $O(\Delta)$ and values in $[O(\Delta)] \cup \{-\infty\}$ (after shifting the indices and values appropriately). Furthermore, note that all convolutions involve monotone non-decreasing sequences. Indeed, as noted above the starting sequence C_0 is monotone non-decreasing. Convolving it with itself produces a monotone non-decreasing sequence again, and the clipping in Line 7 preserves monotonicity. The same argument applies for further iterations.

Algorithm 4 Given an instance $(\mathcal{I}, W)$ of Unbounded Knapsack and a guess $\alpha \in [p_{\max} \cdot W]$, the algorithm computes a value $C_k[W]$ satisfying the guarantee in Lemma 3.11.

```
1  $k := \lceil \log W \rceil$
2  Initialize $C_0[0\,..\,8\Delta] := \mathcal{P}_0[0\,..\,8\Delta]$ by iterating over the item set $\mathcal{I}$ once
3  $J^0 := [W \cdot 2^{-k} - 8\Delta\,..\,W \cdot 2^{-k} + 8\Delta]$
4  for $i = 1, \dots, k$ do
5      $J^i := [W \cdot 2^{i-k} - 8\Delta\,..\,W \cdot 2^{i-k} + 8\Delta]$
6      $A_i[J^i] := \textsc{MaxConv}(C_{i-1}[J^{i-1}], C_{i-1}[J^{i-1}])$
7      For every $j \in J^i$ set $C_i[j] := \begin{cases} \lceil \alpha \cdot 2^{i-k} \rceil + 24\Delta & \text{if } A_i[j] > \alpha \cdot 2^{i-k} + 24\Delta \\ -\infty & \text{if } A_i[j] < \alpha \cdot 2^{i-k} - 40\Delta \\ A_i[j] & \text{otherwise} \end{cases}$
8  return $C_k[W]$
```

The running time of the clipping step in Line 7 takes time $O(\Delta) \leq O(T(\Delta))$. Thus, the running time of Algorithm 4 is $O(n + T(\Delta) \log W)$, where $T(\Delta)$ is the running time to compute Bounded Monotone MaxPlus Conv on sequences of length Δ.

Next we argue about correctness. For each $i \in [k]$, let $J^i = [W \cdot 2^{i-k} - 8\Delta\,..\,W \cdot 2^{i-k} + 8\Delta]$ be as defined in Algorithm 4. The correctness hinges on the following claim.

▷ Claim 3.12. For every $i \in [k]$ and every index $j \in J^i \cap [W]$ the following holds:

- If $\mathcal{P}_i[j] \in [\alpha \cdot 2^{i-k} - 40\Delta\,..\,\alpha \cdot 2^{i-k} + 24\Delta]$, then $C_i[j] = \mathcal{P}_i[j]$.
- If $\mathcal{P}_i[j] > \alpha \cdot 2^{i-k} + 24\Delta$, then $C_i[j] = \lceil \alpha \cdot 2^{i-k} \rceil + 24\Delta$.
- If $\mathcal{P}_i[j] < \alpha \cdot 2^{i-k} - 40\Delta$, then $C_i[j] = -\infty$.

Intuitively, the claim says that entries "close" to the (scaled) guess $\alpha \cdot 2^{i-k}$ get computed exactly, while entries below and above get clipped appropriately.

Proof. We prove the claim by induction on i. For the base case, consider $i = 0$. Note that since $\alpha \in [0\,..\,p_{\max} \cdot W]$ and $k = \lceil \log W \rceil$, we have $\alpha \cdot 2^{-k} \leq p_{\max} \leq \Delta$. Thus, $[\alpha \cdot 2^{-k} - 40\Delta\,..\,\alpha \cdot 2^{-k} + 24\Delta]$ contains the whole interval $[0\,..\,\Delta]$ of possible values of $\mathcal{P}_0[j] = C_0[j]$ (for any $0 \leq j \leq 8\Delta$).

Now we show that the claim holds for any $1 \leq i \leq k$ assuming it holds for $i-1$. Fix any $j \in J^i$. Note that the clipping in Line 7 does not increase any entry, hence $C_i[j] \leq A_i[j]$. Moreover, since inductively $C_{i-1}[j'] \leq \mathcal{P}_{i-1}[j']$ holds for all j', by definition of A_i we have $A_i[j] \leq \mathcal{P}_i[j]$. Hence, we obtain $C_i[j] \leq \mathcal{P}_i[j]$. We use this observation to obtain the claim, by showing an appropriate lower bound for $C_i[j]$ in the following.

Pick indices $j_1 + j_2 = j$ as guaranteed by Lemma 3.10. Note that $j_1, j_2 \in J^{i-1}$. Since in Line 6 we set $A_i[J^i] = \textsc{MaxConv}(C_{i-1}[J^{i-1}], C_{i-1}[J^{i-1}])$, we conclude that $A_i[j] \geq C_{i-1}[j_1] + C_{i-1}[j_2]$.

We proceed by a case distinction on the values of the entries $\mathcal{P}_{i-1}[j_1]$ and $\mathcal{P}_{i-1}[j_2]$:

Case 1: $\mathcal{P}_{i-1}[j_1], \mathcal{P}_{i-1}[j_2] \in [\alpha \cdot 2^{i-1-k} - 40\Delta \,..\, \alpha \cdot 2^{i-1-k} + 24\Delta]$. By the induction hypothesis, both values are computed exactly, that is, $C_{i-1}[j_1] = \mathcal{P}_{i-1}[j_1]$ and $C_{i-1}[j_2] = \mathcal{P}_{i-1}[j_2]$. Thus, $A_i[j] \geq C_{i-1}[j_1] + C_{i-1}[j_2] = \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2] = \mathcal{P}_i[j]$, using property (ii) of Lemma 3.10. Since we observed above that $A_i[j] \leq \mathcal{P}_i[j]$, we obtain $A_i[j] = \mathcal{P}_i[j]$. The clipping in Line 7 yields the claim for this case.

Case 2: $\mathcal{P}_{i-1}[j_1] > \alpha \cdot 2^{i-1-k} + 24\Delta$. Property (iii) of Lemma 3.10 implies that $|\mathcal{P}_{i-1}[j_1] - \mathcal{P}_{i-1}[j_2]| \leq 4\Delta$, and hence $\mathcal{P}_{i-1}[j_2] \geq \alpha \cdot 2^{i-1-k} + 20\Delta$. Thus, property (ii) of Lemma 3.10 implies $\mathcal{P}_i[j] = \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2] > \alpha \cdot 2^{i-k} + 24\Delta$. Therefore, we want to show that $C_i[j] = \lceil \alpha \cdot 2^{i-k} \rceil + 24\Delta$.

By the induction hypothesis, we have $C_{i-1}[j_1] = \lceil \alpha \cdot 2^{i-1-k} \rceil + 24\Delta$ and $C_{i-1}[j_2] \geq \alpha \cdot 2^{i-1-k} + 20\Delta$. Hence, $A_i[j] \geq C_{i-1}[j_1] + C_{i-1}[j_2] > \alpha \cdot 2^{i-k} + 40\Delta$. Due to the clipping in Line 7, we conclude that $C_i[j] = \lceil \alpha \cdot 2^{i-k} \rceil + 24\Delta$, as desired.

Case 3: $\mathcal{P}_{i-1}[j_1] < \alpha \cdot 2^{i-k} - 40\Delta$. Similarly as in case 2, property (iii) of Lemma 3.10 implies that $\mathcal{P}_{i-1}[j_2] \leq \alpha \cdot 2^{i-1-k} - 36\Delta$. Thus, $\mathcal{P}_i[j] = \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2] < \alpha \cdot 2^{i-k} - 76\Delta$. Since $C_i[j] \leq \mathcal{P}_i[j]$, but $C_i[j]$ takes values in $\{-\infty\} \cup [\alpha \cdot 2^{i-k} - 40\Delta \,..\, \alpha \cdot 2^{i-k} + 40\Delta]$ it follows that $C_i[j] = -\infty$. $\triangleleft$

Given the claim, it is easy to see that $C_k[W] \geq \alpha$ if and only if $\mathcal{P}_k[W] \geq \alpha$. Along with the running time analysis argued earlier, we obtain the claimed lemma. $\square$

Given Lemma 3.11, we can do binary search to find the optimal value. This gives an algorithm for Unbounded Knapsack in time $O((n + T(\Delta)) \log W \log \mathrm{OPT})$. To shave the polylog$(W, \mathrm{OPT})$ factors and obtain the running time $\widetilde{O}(n + T(\Delta))$ claimed in Theorem 3.7, we make use of the following lemma. It allows us to reduce the capacity of the instance by repeatedly adding copies of the item with maximum profit-to-weight ratio. Similar *proximity results* have been shown for general ILPs [EW20], for Unbounded Knapsack [Bat+18] and for the Coin Change problem [CH22]. For completeness, we include the proof by Chan and He [CH22, Lemma 4.1].[2]

Lemma 3.13. *Let* $(p_{i^*}, w_{i^*}) := \operatorname{argmax}_{(p,w)\in\mathcal{I}} \frac{p}{w}$. *If* $W \geq 2w_{\max}^3$, *then there exists an optimal solution containing* (p_{i^*}, w_{i^*}).

Proof. Consider an optimal solution x that does not contain item (p_{i^*}, w_{i^*}). If there is an item (p_j, w_j) that appears at least w_{i^*} times in x, then we can replace w_{i^*} of the copies of item (p_j, w_j) by w_j copies of item (p_{i^*}, w_{i^*}). By definition of (p_{i^*}, w_{i^*}), this does not decrease the total profit of the solution, so by optimality of x the new solution x' is also optimal. Therefore, some optimal solution contains (p_{i^*}, w_{i^*}).

It remains to consider the case that x contains less than w_{i^*} copies of every item, so its total weight is at most $n \cdot w_{i^*} \cdot w_{\max}$. Note that $n \leq w_{\max}$, because without loss of

2. Both Chan and He [CH22] and Bateni et al. [Bat+18] show that the same conclusion of the lemma holds if $W > w_{i^*}^2$, with a slightly more involved argument. For our purposes, this simple variant is enough.

generality there is at most one item per distinct weight (otherwise we can keep only the item with the largest profit for each weight). Thus, the total weight of x is at most $w_{\max}^3$. It follows that $W < w_{\max}^3 + w_{i^*} \leq 2w_{\max}^3$, since otherwise we could add at least one copy of (p_{i^*}, w_{i^*}) to x, contradicting its optimality. □

We now put all pieces together to prove Theorem 3.7.

Proof of Theorem 3.7. As a preprocessing step we repeatedly add the item (p_{i^*}, w_{i^*}) to our solution and decrease W by w_{i^*}, as long as $W > 2w_{\max}^3$. This is correct due to Lemma 3.13. After this preprocessing, it holds that $W = O(w_{\max}^3) = O(\Delta^3)$, and thus $\mathrm{OPT} \leq W \cdot p_{\max} = O(\Delta^4)$. Then we do binary search for OPT, using Algorithm 4 as a decision procedure. By Lemma 3.11, the overall running time is $O((n + T(\Delta)) \log^2 \Delta) = \widetilde{O}(n + T(\Delta))$. □

3.2.3 Solution Reconstruction

The algorithm we described gives us the value OPT of the optimal solution. In this section we will describe how to use witness arrays (Lemma 3.5) to reconstruct a feasible solution $x \in \mathbb{N}^n$ such that $p(x) = \mathrm{OPT}$ with only a polylogarithmic overhead in the overall running time.

Lemma 3.14. *An optimal solution x can be reconstructed in time $\widetilde{O}(n + T(p_{\max} + w_{\max}))$.*

Proof Sketch. Let $k = \lceil \log W \rceil$ be as in Algorithm 4. After determining the value of OPT, run Algorithm 4 again with the guess $\alpha = \mathrm{OPT}$. For every max-plus convolution computed in Line 6, we additionally compute the witness array M_i corresponding to A_i via Lemma 3.5. This takes time $\widetilde{O}(n + T(p_{\max} + w_{\max}))$. Now, the idea is to start from $C_k[W]$ and traverse the *computation tree* of Algorithm 4 backwards. That is, we look at the pair of entries $C_{k-1}[M_k[W]], C_{k-1}[W - M_k[W]]$ which define the value of $C_k[W]$ and recursively obtain the pair of entries in C_{k-2} determining the value of $C_{k-1}[M_k[W]]$, etc. By proceeding in this way, we eventually hit the leaves, i.e., the entries of $C_0[0\,..\,8\Delta] = \mathcal{P}_0[0\,..\,8\Delta]$, which correspond to the items in an optimal solution. A naive implementation of this idea takes time $O(\sum_{i \leq k} 2^i) = O(2^k) = O(W)$, which is too slow for us.

Now we describe an efficient implementation of the same idea. For each $i \in [k]$ construct an array $Z_i[W \cdot 2^{i-k} - 8\Delta\,..\,W \cdot 2^{i-k} + 8\Delta]$ initialized to zeros. Set $Z_k[W] := 1$. We will maintain the invariant that $Z_i[j]$ stores the number of times we arrive at $C_i[j]$ by traversing the computation tree starting at $C_k[W]$. This clearly holds for $Z_k[W] = 1$ by definition. Now we describe how to fill the entries for the levels below. Iterate over $i = k, k-1, \dots, 1$. For each entry $j \in [W \cdot 2^{i-k} - 8\Delta\,..\,W \cdot 2^{i-k} + 8\Delta] \cap [W]$ add $Z_i[j]$ to its witness entries in the level below, i.e., increase $Z_{i-1}[M_i[j]]$ by $Z_i[j]$ and $Z_{i-1}[j - M_i[j]]$ by $Z_i[j]$. The invariant is maintained by definition of the witnesses, and because Algorithm 4 guarantees that $M_i[j], j - M_i[j] \in [W \cdot 2^{i-1-k} - 8\Delta\,..\,W \cdot 2^{i-1-k} + 8\Delta] \cap [W]$. Note that this procedure takes time $O(k\Delta) = \widetilde{O}(\Delta)$.

Finally, note that for the base case we have that each $Z_0[j]$ for $j \in [8\Delta]$ counts the number of times that we hit the entry $C_0[j] = \mathcal{P}_0[j]$ in the computation tree starting from $C_k[W]$. Recall that by definition, $\mathcal{P}_0[j]$ is the maximum profit of an item in $\mathcal{I}$ with weight at most j. Hence, every entry $\mathcal{P}_0[j]$ corresponds to a unique item in $\mathcal{I}$. Therefore, we can read off from $Z_0[0\,..\,8\Delta]$ the multiplicity of each item included in an optimal solution. The overall time of the procedure is $\widetilde{O}(n + T(p_{\max} + w_{\max}))$, as claimed. □

3.3 Exact Algorithm for Knapsack

In this section we prove the following reduction:

Theorem 3.15. *If Bounded Monotone MaxPlus Conv on length-n sequences can be solved in time $T(n)$, then Knapsack can be solved in time $\widetilde{O}(n + T(W + \mathrm{OPT}))$ with high probability.*

Observe that Main Theorem 1.1 follows immediately, by combining Theorem 3.15 with the algorithm for Bounded Monotone MaxPlus Conv of Chi et al. (Corollary 3.3).

Our starting point is the following result of Cygan, Mucha, Węgrzycki and Włodarczyk [Cyg+19], showing a reduction from Knapsack to Bounded Monotone MaxPlus Conv:

Theorem 3.16 ([Cyg+19, Theorem 13])**.** *If the max-plus convolution of length-n sequences can be computed in time $T(n)$, then Knapsack can be solved in time*

$$O(T(W \log W) \log^3(n/\delta) \log n)$$

with probability at least $1 - \delta$.

Their reduction is a generalization of Bringmann's algorithm for Subset Sum [Bri17], which can be seen as a reduction from Subset Sum to Boolean convolution. Cygan et al. showed that the reduction for Knapsack can be obtained by essentially replacing the Boolean convolutions by max-plus convolutions in Bringmann's algorithm.

For our purposes, we observe that essentially the same reduction yields sequences of length $O(W)$ which are monotone and have entries bounded by OPT. In particular, these are Bounded Monotone MaxPlus Conv instances.

Proof of Theorem 3.15. The proof is virtually the same as [Cyg+19, Theorem 13], so we omit some details. In particular, we emphasize how the constructed instances can be seen to be monotone and bounded, but we omit some details of the correctness argument.

Given the item set $\mathcal{I}$, we work with the profit sequence $\mathcal{P}_{\mathcal{I}}[\cdot]$, where for each $j \in \mathbb{N}$ we have

$$\mathcal{P}_{\mathcal{I}}[j] = \max\{\, p_{\mathcal{I}}(x) \mid x \in \{0, 1\}^n, w_{\mathcal{I}}(x) \le j \,\}.$$

Our goal will be to compute $\mathrm{OPT} = \mathcal{P}_{\mathcal{I}}[W]$.

The idea of the algorithm is the following: split the item set $\mathcal{I}$ into groups $G_{(a,b)} \subseteq \mathcal{I}$ such that all items $(p, w) \in \mathcal{I}$ with $2^{a-1} \leq p < 2^a$ and $2^{b-1} \leq w < 2^b$ are in group $G_{(a,b)}$. That is, all items within each group have weights and profits within a factor of 2 of each other, and thus there are $O(\log W \log \mathrm{OPT})$ many groups. We will describe how to compute $\mathcal{P}_{G_{(a,b)}}[0\,..\,W]$ for each $G_{(a,b)}$. Having that, we simply combine all the profit arrays into $\mathcal{P}_{\mathcal{I}}[0\,..\,W]$ using max-plus convolutions. Since we have $O(\log W \log \mathrm{OPT})$ groups, and each profit array is a monotone non-decreasing sequence of length W with entries bounded by OPT, the combination step takes time $\widetilde{O}(T(W + \mathrm{OPT}))$.

Fix some group $G_{(a,b)}$. Since every $(p, w) \in G_{(a,b)}$ has $w \geq 2^{a-1}$ and $p \geq 2^{b-1}$, any feasible solution from $G_{(a,b)}$ consists of at most $z := \lceil \min\{ W/2^{a-1}, \mathrm{OPT}/2^{b-1} \} \rceil$ items. Thus, by splitting the items in $G_{(a,b)}$ randomly into z subgroups $G_{(a,b),1}, \ldots, G_{(a,b),z}$, any fixed feasible solution has at most $O(\log z)$ items in each subgroup $G_{(a,b),k}$ with high probability. To see this, fix a solution x and note that,

$$\mathbb{P}[\text{at least } r \text{ items from } x \text{ fall in } G_{(a,b),k}] \leq \binom{z}{r}\left(\frac{1}{z}\right)^r \leq \left(\frac{e \cdot z}{r}\right)^r \left(\frac{1}{z}\right)^r = \left(\frac{e}{r}\right)^r,$$

where the first inequality follows due to a union bound over all subsets of items of size r. By setting $r = O(\log z)$, we can bound this probability by z^{-c} for any constant c. So by a union bound, none of the z groups $G_{(a,b),1}, \ldots, G_{(a,b),z}$ has more than $\kappa := O(\log z)$ elements from the fixed solution x with probability at least $1 - 1/\mathrm{poly}(z)$.

Therefore, to obtain the value of any fixed solution it suffices to compute the optimal solution consisting of at most κ items from $G_{(a,b),k}$ for every target weight $\leq O(2^a\kappa)$, and then merge the results. More precisely, for every $1 \leq i \leq z$ we compute the array $\mathcal{P}_{G_{(a,b),i},\kappa}[0\,..\,O(2^a\kappa)]$ defined as

$$\mathcal{P}_{G_{(a,b),i},\kappa}[j] := \max\{\, p(x) \mid x \text{ is a solution from } G_{(a,b),i} \text{ with } w(x) \leq j, \|x\|_1 \leq \kappa \,\}$$

for each $j \in [O(2^a\kappa)]$. For ease of notation, we denote the array by $\mathcal{P}_{G_i,\kappa} := \mathcal{P}_{G_{(a,b),i},\kappa}$.

Then, we merge the $\mathcal{P}_{G_i,\kappa}$'s using max-plus convolutions to obtain $\mathcal{P}_{G_{(a,b)}}$. Now we describe these two steps in more detail:

Computing $\mathcal{P}_{G_i,\kappa}[0\,..\,O(2^a\kappa)]$ Since we only care about solutions with at most κ items, we use randomization again[3]: split the items in $G_{(a,b),i}$ into κ^2 buckets $A_1, \ldots, A_{\kappa^2}$. By the birthday paradox, with constant probability it holds that any fixed solution is shattered among the buckets, i.e., each bucket contains at most 1 item of the solution. Thus, for each bucket A_k we construct the array $\mathcal{P}_{A_k,1}[0\,..\,2^a]$. This is defined as above, namely,

$$\mathcal{P}_{A_k,1}[j] := \max\{\, p(x) \mid x \text{ is a solution from } A_k \text{ with } w(x) \leq j, \|x\|_1 \leq 1 \,\}$$

for each entry $j \in [2^a]$. Then, we merge $\mathcal{P}_{A_1,1}, \mathcal{P}_{A_2,1}, \ldots, \mathcal{P}_{A_{\kappa^2},1}$ using max-plus convolutions. By definition, every $\mathcal{P}_{A_i,1}$ is a monotone non-decreasing sequence of length 2^a with entries bounded by 2^b. Thus, the merging step takes time $O(T((2^a + 2^b) \cdot \kappa^2) \cdot \kappa^2)$.

3. This step is called "Color Coding" in [Bri17; Cyg+19].

Each entry of the resulting array has the correct value $\mathcal{P}_{G_i,\kappa}[j]$ with constant probability, since a corresponding optimal solution is shattered with constant probability. By repeating this process $O(\log z)$ times and keeping the entrywise maximum among all repetitions, we boost the success probability to $1 - 1/\mathrm{poly}(z)$. Thus, by a union bound over the z subgroups $G_{(a,b),1}, \dots, G_{(a,b),z}$, we get that any z fixed entries $\mathcal{P}_{G_1,\kappa}[j_1], \dots, \mathcal{P}_{G_z,\kappa}[j_z]$ corresponding to a solution which is partitioned among the z subgroups get computed correctly with probability at least $1 - 1/\mathrm{poly}(z)$. This adds an extra $O(\log z) = O(\kappa)$ factor to the running time.

Merging $\mathcal{P}_{G_1,\kappa}, \dots, \mathcal{P}_{G_z,\kappa}$ This computation is done in a binary tree-like fashion. That is, in the first level we compute

$$\textsc{MaxConv}(\mathcal{P}_{G_1,\kappa}, \mathcal{P}_{G_2,\kappa}), \textsc{MaxConv}(\mathcal{P}_{G_3,\kappa}, \mathcal{P}_{G_4,\kappa}), \dots, \textsc{MaxConv}(\mathcal{P}_{G_{z-1},\kappa}, \mathcal{P}_{G_z,\kappa}).$$

In the second level we merge the results from the first level in a similar way. We proceed in the same way with further levels. Since we merge z sequences, we have $\lceil \log z \rceil$ levels of computation. In the j-th level, we compute the max-plus convolution of $z/2^j$ many monotone non-decreasing sequences of length $O(2^j \cdot 2^a \cdot \kappa)$ with entries bounded by $O(2^j \cdot 2^b \cdot \kappa)$. Therefore, overall the merging takes time

$$O\left(\sum_{j=1}^{\lceil \log z \rceil} \frac{z}{2^j} \cdot T((2^a + 2^b) \cdot 2^j \cdot \kappa) \right) \le \widetilde{O}(T((2^a + 2^b) \cdot z)),$$

where we used both points in Assumption 3.6, namely, that $k \cdot T(n) \le O(T(k \cdot n))$ for any $k > 1$ and that $T(\widetilde{O}(n)) \le \widetilde{O}(T(n))$. Since $z = \lceil \min\{ W/2^{a-1}, \mathrm{OPT}/2^{b-1} \} \rceil$, we can bound this running time as

$$\widetilde{O}(T((2^a + 2^b) \cdot z) = \widetilde{O}(T(W + \mathrm{OPT})).$$

Wrapping up To recap, the algorithm does the following steps:

1. Split the items into $O(\log W \log \mathrm{OPT})$ groups $G_{(a,b)}$. This takes time $\widetilde{O}(n)$.
2. Randomly split each group $G_{(a,b)}$ into $z := \lceil \min\{ W/2^{a-1}, \mathrm{OPT}/2^{b-1} \} \rceil$ subgroups $G_{(a,b),i}$ for $i \in [z]$.
3. For each $G_{(a,b),i}$ compute the array $\mathcal{P}_{G_i,\kappa}[0\,..\,O(2^a\kappa)]$ in time $O(T((2^a + 2^b)\kappa^2) \cdot \kappa^2)$. Since $\kappa = O(\log z)$, the total time over all $i \in [z]$ is

$$\begin{aligned} O(z \cdot T((2^a + 2^b)\kappa^2) \cdot \kappa^2) \le O(T((2^a + 2^b) \cdot z \cdot \kappa^2)\kappa^2) &\le \widetilde{O}(T((2^a + 2^b) \cdot z)) \\ &\le \widetilde{O}(T(W + \mathrm{OPT})). \end{aligned}$$

   Note that here we used the niceness Assumption 3.6 on $T(n)$. In particular, first we used that $k \cdot T(n) \le O(T(k \cdot n))$ for any $k > 1$, and then that $T(\widetilde{O}(n)) \le \widetilde{O}(T(n))$.

4. Merge the arrays $\mathcal{P}_{G_{1,\kappa}} \dots \mathcal{P}_{G_{z,\kappa}}$ using max-plus convolutions in a tree-like fashion in total time $\widetilde{O}(T(W + \mathrm{OPT}))$ to obtain $\mathcal{P}_{G_{(a,b)}}[0\,..\,W]$.

5. Merge the arrays $\mathcal{P}_{G_{(a,b)}}$ using $O(\log W \log \mathrm{OPT})$ max-plus convolutions in total time $\widetilde{O}(T(W + \mathrm{OPT}))$.

Thus, the overall time of the algorithm is $\widetilde{O}(n + T(W + \mathrm{OPT}))$. Note that as mentioned earlier in the proof, the algorithm succeeds in computing any *fixed* entry $\mathcal{P}_{\mathcal{I}}[j]$ with probability at least $1 - 1/\mathrm{poly}(z)$. In particular, this is sufficient to compute the optimal solution $\mathcal{P}_{\mathcal{I}}[W]$ with good probability.

As described, the algorithm only returns the value of the optimal solution. We can easily reconstruct an optimal solution $x \in \mathbb{N}^n$ for which $p(x) = \mathrm{OPT}$, as we sketch now. Note that at the end of the algorithm, we will have a sequence whose entries correspond to $\mathcal{P}[0\,..\,W]$. This sequence was obtained as the max-plus convolution of two distinct other sequences, call them $A[0\,..\,W], B[0\,..\,W]$. For the output entry $\mathcal{P}[W]$, we can find its witness $i \in [W]$, i.e. the index i such that $\mathcal{P}[W] = A[i] + B[W - i]$. Note that we can find i in time $O(W)$ by simply trying all possibilities. Then, we continue recursively finding the witnesses for i and for $W - i$. Eventually, we will reach the entries of the arrays $\mathcal{P}_{A_k,0}$ which correspond to single items from $\mathcal{I}$, and these form the solution x. The crucial observation is that because the algorithm never convolves a sequence with itself (unlike our algorithm in Theorem 3.7 for Unbounded Knapsack), this recursive process finds at most one witness per convolution. Hence, the total time spent is proportional to the total length of the convolved sequences $\widetilde{O}(W)$. □

3.4 Approximation Schemes for Unbounded Knapsack

In this section we turn to approximation schemes for Unbounded Knapsack. Our goal is to prove Theorem 1.5 and Main Theorem 1.6, which we restate for convenience.

Theorem 1.5. *Unbounded Knapsack has a deterministic FPTAS that runs in time*

$$\widetilde{O}\left(n + \frac{(1/\varepsilon)^2}{2^{\Omega(\sqrt{\log(1/\varepsilon)})}}\right).$$

Main Theorem 1.6. *Unbounded Knapsack has a weak approximation scheme running in expected time $\widetilde{O}(n + (\frac{1}{\varepsilon})^{1.5})$.*

Before diving in the details, we start with a high level technical overview.

Proof Overview Let $(\mathcal{I}, W)$ be an instance of Unbounded Knapsack. The starting point is the following "repeated squaring" approach. Namely, recall that we denote by $\mathcal{P}_i[0\,..\,W]$ the sequence where $\mathcal{P}_i[j]$ is the maximum profit of any solution of weight at most j and with at most 2^i items. Since $W/w_{\min}$ is an upper bound on the number of items in the optimal solution, we have that $\mathcal{P}_{\log(W/w_{\min})}[W] = \mathrm{OPT}$. We can compute

$\mathcal{P}_i[0\,..\,W]$ by taking the max-plus convolution of $\mathcal{P}_{i-1}[0\,..\,W]$ with itself. This yields an algorithm in time $O(W^2 \log W/w_{\min})$.

Although this exact algorithm is not particularly exciting or new, it can be nicely extended to the approximate setting. In Section 3.4.2 we show that by replacing the exact max-plus convolutions with approximate ones, we obtain an FPTAS for Unbounded Knapsack in time $\widetilde{O}(n + 1/\varepsilon^2)$. To this end, we preprocess the item set to get rid of *light* items with weight smaller than $\varepsilon \cdot W$, and *cheap* items with profit smaller than $\varepsilon \cdot \mathrm{OPT}$, while decreasing the optimal value by only $O(\varepsilon \cdot \mathrm{OPT})$ (see Lemma 3.20). After this preprocessing, we have that the maximum number of items in any solution is $W/w_{\min} < 1/\varepsilon$. Thus, we only need to approximate $O(\log 1/\varepsilon)$ convolutions. For this we use an algorithm due to Chan [Cha18], which in our setting without cheap items runs in time $\widetilde{O}(1/\varepsilon^2)$ (see Lemma 3.22). Thus, after applying the preprocessing in time $O(n)$, we compute a $(1+\varepsilon)$-approximation of $\mathcal{P}[0\,..\,W]$ by applying $O(\log 1/\varepsilon)$ approximate max-plus convolutions in overall time $\widetilde{O}(n + (1/\varepsilon)^2)$. , yielding Theorem 1.5.

In Section 3.4.3 we treat the case of weak approximation. The main steps of the algorithm are virtually the same as before. The crucial difference is that now we can afford to round weights. In this way, we can adapt Chan's algorithm and construct max-plus convolution instances which are monotone non-decreasing and have bounded entries (see Lemma 3.27). This yields Bounded Monotone MaxPlus Conv instances, and by using Chi, Duan, Xie and Zhang's algorithm for this special case [Chi+22], we can compute a weak approximation of max-plus convolution in time $\widetilde{O}((1/\varepsilon)^{1.5})$. By similar arguments as for the strong approximation, this yields a weak approximation scheme running in time $\widetilde{O}(n + (1/\varepsilon)^{1.5})$, proving Main Theorem 1.6.

3.4.1 Preparations

Notions of Approximation Here formalize the notions of approximation that we will be working with. We say that an algorithm gives a *strong* $(1+\varepsilon)$-approximation for Unbounded Knapsack if it returns a solution $x \in \mathbb{N}^n$ with weight $w(x) \leq W$ and profit $p(x) \geq (1-\varepsilon) \cdot \mathrm{OPT}$. An algorithm gives a *weak* $(1+\varepsilon)$-approximation for Unbounded Knapsack if it returns a solution $x \in \mathbb{N}^n$ with profit $p(x) \geq (1-\varepsilon) \cdot \mathrm{OPT}$ and weight $w(x) \leq (1+\varepsilon) \cdot W$. We stress that here OPT still denotes the optimum value with weight at most W, i.e., $\mathrm{OPT} = \max\{\, p(x) \mid x \in \mathbb{N}^n,\, w(x) \leq W \,\}$.

Greedy 2-approximation The fractional solution for an instance $(\mathcal{I}, W)$ of Unbounded Knapsack has a simple structure: pack the entire capacity W greedily with the *most efficient item* i^*. That is, choose the item $(p_{i^*}, w_{i^*}) \in \mathcal{I}$ which maximizes the ratio p_{i^*}/w_{i^*} and add it W/w_{i^*} many times. Since $\lfloor W/w_{i^*} \rfloor$ copies forms a feasible integral solution, it holds that $\lfloor W/w_{i^*} \rfloor \cdot p_{i^*} \leq \mathrm{OPT} \leq (W/w_{i^*}) \cdot p_{i^*}$. By $W/w_{i^*} \geq 1$, we have $\lfloor W/w_{i^*} \rfloor \geq 1/2 \cdot W/w_{i^*}$. Thus, the greedy solution is a 2-approximation to OPT. Note that we can find this solution in time $O(n)$.

Profit Sequence Similarly as in Section 3.2, given an item set $\mathcal{I}$ we work with the profit sequence $\mathcal{P}_{\mathcal{I}}[\cdot]$. For $j \in \mathbb{N}$, this is defined as

$$\mathcal{P}_{\mathcal{I}}[j] := \max\{\, p(x) \mid w(x) \leq j \,\}.$$

Moreover, for $k \geq 0$ we define $\mathcal{P}_{\mathcal{I},k}[\cdot]$ where for $j \in \mathbb{N}$ we set

$$\mathcal{P}_{\mathcal{I},k} := \max\{\, p(x) \mid w(x) \leq j, \|x\|_1 \leq 2^i \,\}.$$

When the item set $\mathcal{I}$ is clear from context, we drop it and write $\mathcal{P}[\cdot]$ and $\mathcal{P}_k[\cdot]$.

Connection to max-plus convolution The following lemma shows that if we know $\mathcal{P}_{i-1}$, we can compute $\mathcal{P}_i$ by applying a max-plus convolution of $\mathcal{P}_{i-1}$ with itself.

Lemma 3.17 (Halving Lemma). *For any $i \geq 1$ and $w \geq 0$, it holds that*

$$\mathcal{P}_i[0\,..\,W] = \textsc{MaxConv}(\mathcal{P}_{i-1}[0\,..\,W], \mathcal{P}_{i-1}[0\,..\,W]).$$

Proof. We denote the right hand side by $C := \textsc{MaxConv}(\mathcal{P}_{i-1}[0\,..\,W], \mathcal{P}_{i-1}[0\,..\,W])$. Fix some $j \in [w]$. We will show that $\mathcal{P}_i[j] = C[j]$:

$\mathcal{P}_i[j] \geq C[j]$: This holds since $C[j]$ corresponds to the profit of some solution with at most 2^i items, but by definition $\mathcal{P}_i[j]$ is the maximum profit among all such solutions.

$\mathcal{P}_i[j] \leq C[j]$: Let $x \in \mathbb{N}^n$ be the solution corresponding to $\mathcal{P}_i[j]$, i.e. such that $p(x) = \mathcal{P}_i[j]$, $w(x) \leq j$ and $\|x\|_1 \leq 2^i$. Split x in two solutions $x = x_1 + x_2$ of roughly the same size $\|x_1\| \approx \|x_2\|_1 \approx \|x\|_1/2$ (if $\|x\|_1$ is odd, put an extra item to x_1 or x_2). In this way, it holds that both x_1 and x_2 are solutions of at most 2^{i-1} items and weight at most j and therefore by the optimality of $\mathcal{P}_{i-1}$ it holds that $p(x_1) \leq \mathcal{P}_{i-1}[w(x_1)]$ and $p(x_2) \leq \mathcal{P}_{i-1}[w(x_2)]$. Hence, by definition of C we conclude that $C[j] \geq \mathcal{P}_{i-1}[j - w(x_2)] + \mathcal{P}_{i-1}[w(x_2)] \geq p(x_1) + p(x_2) = \mathcal{P}_i[j]$. □

Since any item has weight at least 1, the optimal solution consists of at most W items. Thus, we can use Lemma 3.17 to compute $\mathrm{OPT} = \mathcal{P}_{\lceil \log W \rceil}[W]$ by applying $O(\log W)$ max-plus convolutions (note that the base case $\mathcal{P}_0[0\,..\,W]$ corresponds to solutions of at most one item, and thus can be computed easily). Our approximation schemes will implement this "repeated squaring" algorithm by appropriately approximating the max-plus convolutions.

3.4.2 A simplified FPTAS

In this section we prove the following reduction from approximating Unbounded Knapsack to max-plus convolution.

Theorem 3.18. *If max-plus convolution can be solved in time $T(n)$, then Unbounded Knapsack has an FPTAS in time $\widetilde{O}(n + T(1/\varepsilon))$.*

This proves Theorem 1.5 by plugging in the deterministic bound $n^2/2^{\Omega(\sqrt{\log n})}$ for max-plus convolution [Bre+14; CW21].

Preprocessing

To give the FPTAS we start with some preprocessing to remove items with small profit and items with small weight.

Step 1: Remove cheap items We first remove items with small profit. Let P_0 be the value of the greedy 2-approximation and set $T := 2\varepsilon P_0$. Split the items into *expensive* $\mathcal{I}_E := \{ (p,w) \in \mathcal{I} \mid p > T \}$, and *cheap* $\mathcal{I}_C := \mathcal{I} \setminus \mathcal{I}_E$. Let $i^* := \operatorname{argmax}_{i \in \mathcal{I}_C} \frac{p_i}{w_i}$ be the item corresponding to the greedy 2-approximation for the cheap items and set $r := \lceil T/p_{i^*} \rceil$. We delete all cheap items from $\mathcal{I}$ and add an item corresponding to r copies of (p_{i^*}, w_{i^*}), i.e., we set $\mathcal{I}' := \mathcal{I}_E \cup \{ (r \cdot p_{i^*}, r \cdot w_{i^*}) \}$. Note that now any item in $\mathcal{I}'$ has profit at least T. These steps are summarized in Algorithm 5. The following lemma shows that this decreases the total profit of any solution by at most $O(\varepsilon \cdot \mathrm{OPT})$.

Algorithm 5 PREPROCESSING-PROFITS($\mathcal{I}, W$): Preprocessing to remove low profit items.

```
1   Let $P_0$ be the value of the greedy 2-approximation of $(\mathcal{I}, W)$
2   $T := 2 \cdot \varepsilon \cdot P_0$
3   $\mathcal{I}_E := \{ (p,w) \in \mathcal{I} \mid p > T \}, \mathcal{I}_C := \mathcal{I} \setminus \mathcal{I}_E$
4   $i^* := \operatorname{argmax}_{i \in \mathcal{I}_C} \frac{p_i}{w_i}$
5   $r := \lceil T/p_{i^*} \rceil$
6   return $\mathcal{I}_E \cup \{ (r \cdot p_{i^*}, r \cdot w_{i^*}) \}$
```

Lemma 3.19 (Preprocessing Profits). *Let $\mathcal{I}'$ be the result of running Algorithm 5 on $(\mathcal{I}, W)$. Then, for every $w \in [W]$, it holds that $\mathcal{P}_{\mathcal{I}'}[w] \geq \mathcal{P}_{\mathcal{I}}[w] - 4\varepsilon \cdot \mathrm{OPT}$. Moreover, the minimum profit in $\mathcal{I}'$ is $p_{\min} > \varepsilon\mathrm{OPT}$.*

Proof. Fix some weight $w \in [W]$ and let $x \in \mathbb{N}^{|\mathcal{I}|}$ be the solution from $\mathcal{I}$ corresponding to the entry $\mathcal{P}_{\mathcal{I}}[w]$. We denote by x_C the *cheap* items in x, i.e., those with profit $\leq T = 2\varepsilon P_0$. Thus, we can write $w_{\mathcal{I}}(x) = w_{\mathcal{I}}(x_C) + w_{\mathcal{I}}(x - x_C)$ and similarly $p_{\mathcal{I}}(x) = p_{\mathcal{I}}(x_C) + p_{\mathcal{I}}(x - x_C)$.

Now we construct a solution $x' \in \mathbb{N}^{|\mathcal{I}'|}$ from the preprocessed $\mathcal{I}'$ which will satisfy $w_{\mathcal{I}'}(x') \leq w$ and $p_{\mathcal{I}'}(x') \geq p - \varepsilon \cdot \mathrm{OPT}$, yielding the lemma. We start from x and remove all the cheap items x_C, replacing them by $\lfloor w_{\mathcal{I}}(x_C)/(r \cdot w_{i^*}) \rfloor$-many copies of $(r \cdot p_{i^*}, r \cdot w_{i^*}) \in \mathcal{I}'$. Note that in this way, $w_{\mathcal{I}'}(x') \leq w$. Now we lower bound the profit. Since (p_{i^*}, w_{i^*}) is the item corresponding to the greedy 2-approximation from the cheap items, it follows that $p_{\mathcal{I}}(x_C) \leq \frac{w_{\mathcal{I}}(x_C)}{w_{i^*}} \cdot p_{i^*}$. Further, by definition of r it holds that $r \leq T/p_{i^*} + 1$. Thus,

$$\begin{aligned} p_{\mathcal{I}'}(x') &= p_{\mathcal{I}}(x - x_C) + r \cdot p_{i^*} \cdot \lfloor w(x_C)/(r \cdot w_{i^*}) \rfloor \\ &\geq p_{\mathcal{I}}(x - x_C) + p_{\mathcal{I}}(x_C) - r \cdot p_{i^*} \geq p_{\mathcal{I}}(x) - T - p_{i^*}. \end{aligned}$$

Since $p_{i^*} \leq T = 2\varepsilon \cdot P_0$, we conclude that $p_{\mathcal{I}'}(x') \geq p_{\mathcal{I}}(x) - 4\varepsilon \cdot P_0$. Therefore, $\mathcal{P}_{\mathcal{I}'}[w] \geq p_{\mathcal{I}'}(x') \geq p_{\mathcal{I}}(x) - 4\varepsilon \cdot P_0 \geq \mathcal{P}_{\mathcal{I}}[w] - 4\varepsilon \cdot \mathrm{OPT}$, as desired.

Finally, note that by construction we have that $p_{\min} > 2\varepsilon P_0 \geq \varepsilon\mathrm{OPT}$. □

Step 2: Remove light items Having removed all items with low profit, we proceed similarly to remove items of low weight. More precisely, we say that an item is light if its weight is less than $\varepsilon \cdot W$. We remove all light items except for the most profitable one (i.e. the one with the best profit to weight ratio), which we *copy* enough times to make it have weight at least $\varepsilon \cdot W$. The details are shown in Algorithm 6.

Algorithm 6 PREPROCESSING-PROFITS-AND-WEIGHTS$(\mathcal{I}, W)$: Preprocessing to remove items of low profit and low weight.

```
1  $\widetilde{\mathcal{I}} := $ PREPROCESSING-PROFITS$(\mathcal{I}, W)$ (Algorithm 5)
2  $\mathcal{I}_L := \{ (p,w) \in \widetilde{\mathcal{I}} \mid w < \varepsilon \cdot W \}, \mathcal{I}_H := \widetilde{\mathcal{I}} \setminus \mathcal{I}_S$
3  $i^* := \mathrm{argmax}_{i \in \mathcal{I}_L} \frac{p_i}{w_i}$
4  $r := \lceil \frac{\varepsilon \cdot W}{w_{i^*}} \rceil$
5  return $\mathcal{I}_H \cup \{ (r \cdot p_{i^*}, r \cdot w_{i^*}) \}$
```

The following lemma shows that removing cheap and light items as in Algorithm 6 only decreases the profit of any solution by $O(\varepsilon\mathrm{OPT})$.

Lemma 3.20 (Preprocessing Profits and Weights)**.** *Let $\mathcal{I}'$ be the result of running Algorithm 6 on $(\mathcal{I}, W)$. Then any solution for $\mathcal{I}'$ has a corresponding solution for $\mathcal{I}$ with the same profit and weight. Further, for any $w \in [W]$ it holds that $\mathcal{P}_{\mathcal{I}'}[w] \geq \mathcal{P}_{\mathcal{I}}[w] - 8\varepsilon \cdot \mathrm{OPT}$. Moreover, the minimum profit and minimum weight in $\mathcal{I}'$ satisfy $p_{\min} > \varepsilon\mathrm{OPT}$ and $w_{\min} > \varepsilon W$.*

Proof. Fix $w \in [W]$ and $p := \mathcal{P}_{\mathcal{I}}[w]$. Let $\widetilde{\mathcal{I}}$ be the item set obtained in Line 1. Lemma 3.19 guarantees that there is a solution x from $\widetilde{\mathcal{I}}$ with $p_{\widetilde{\mathcal{I}}}(x) \geq p - 4\varepsilon \cdot \mathrm{OPT}$ and $w_{\widetilde{\mathcal{I}}}(x) \leq w$. Let x_L be the light items in x, i.e., those with weight $< \varepsilon \cdot W$. We construct a solution x' of items from $\mathcal{I}'$ by taking x, removing all the items in x_L and replacing them by $\lfloor w_{\widetilde{\mathcal{I}}}(x_L)/(r \cdot w_{i^*}) \rfloor$-many copies of $(r \cdot p_{i^*}, r \cdot w_{i^*})$. Then, the total profit of x' is

$$\begin{aligned} p_{\mathcal{I}'}(x') &= p_{\widetilde{\mathcal{I}}}(x - x_L) + \left\lfloor \frac{w_{\widetilde{\mathcal{I}}}(x_L)}{r \cdot w_{i^*}} \right\rfloor \cdot r \cdot p_{i^*} \geq p_{\widetilde{\mathcal{I}}}(x - x_L) + \frac{w_{\widetilde{\mathcal{I}}}(x_L)}{r \cdot w_{i^*}} \cdot r \cdot p_{i^*} - r \cdot p_{i^*} \\ &\geq p_{\widetilde{\mathcal{I}}}(x - x_L) + p_{\widetilde{\mathcal{I}}}(x_L) - r \cdot p_{i^*} = p_{\widetilde{\mathcal{I}}}(x) - r \cdot p_{i^*}, \end{aligned} \tag{3.2}$$

where the second inequality $p(x_L) \leq \frac{w(x_L)}{r \cdot w_{i^*}} \cdot r \cdot p_{i^*}$ follows since by definition (p_{i^*}, w_{i^*}) is the most efficient of the light items. Now we argue that we can bound the last term $p_{i^*} \cdot r$ by $O(\varepsilon\mathrm{OPT})$. As w_{i^*} is a light item, we have $w_{i^*} \leq \varepsilon W$. Since for any number $z \geq 1$ we have that $\lceil z \rceil \leq 2z$, it follows that $r = \lceil (\varepsilon W)/w_{i^*} \rceil \leq 2\varepsilon W/w_{i^*}$. Since packing $\lfloor W/w_{i^*} \rfloor$ copies of item i^* is a feasible solution, it holds that $\mathrm{OPT} \geq \lfloor W/w_{i^*} \rfloor p_{i^*} \geq \frac{W p_{i^*}}{2w_{i^*}}$. Combining both inequalities we get that $r \cdot p_{i^*} \leq 2\varepsilon W p_{i^*}/w_{i^*} \leq 4\varepsilon \cdot \mathrm{OPT}$. Therefore, by (3.2) we conclude that $p_{\mathcal{I}'}(x') \geq p_{\widetilde{\mathcal{I}}}(x) - r \cdot p_{i^*} \geq p_{\widetilde{\mathcal{I}}}(x) - 4\varepsilon \cdot \mathrm{OPT} \geq p - 8\varepsilon \cdot \mathrm{OPT}$.

Similarly, we can bound the total weight of x' as

$$w_{\mathcal{I}'}(x') = w_{\widetilde{\mathcal{I}}}(x - x_L) + \left\lfloor \frac{w_{\widetilde{\mathcal{I}}}(x_L)}{r \cdot w_{i^*}} \right\rfloor \cdot r \cdot w_{i^*} \leq w_{\widetilde{\mathcal{I}}}(x - x_L) + w_{\widetilde{\mathcal{I}}}(x_L) \leq w_{\widetilde{\mathcal{I}}}(x) \leq w.$$

Finally, note that $p_{\min} > \varepsilon \mathrm{OPT}$ follows directly from Lemma 3.19 and $w_{\min} > \varepsilon W$ by construction. □

The Algorithm

To obtain the FPTAS, we first preprocess the set of items $\mathcal{I}$ using Algorithm 6 and obtain a set $\mathcal{I}'$. Then, by Lemma 3.20, it holds that $\mathcal{P}_{\mathcal{I}'}[W] \geq (1 - O(\varepsilon)) \cdot \mathrm{OPT}$, so to obtain a $(1 + O(\varepsilon))$-approximation of $\mathcal{P}_{\mathcal{I}}[W]$ it suffices to give a $(1 + \varepsilon)$-approximation of $\mathcal{P}_{\mathcal{I}'}[W]$.

In fact, we solve a slightly more general problem, and compute a sequence which gives a (pointwise) $(1 + \varepsilon)$-approximation of the entire sequence $\mathcal{P}_{\mathcal{I}'}[0\,..\,W]$, according to the following definition.

Definition 3.21 (Strong Approximation of a Sequence). *We say that a sequence $A[0\,..\,n]$ (pointwise) $(1 + \varepsilon)$-approximates a sequence $B[0\,..\,n]$ if $B[i]/(1 + \varepsilon) \leq A[i] \leq B[i]$ for every $i \in [n]$.*

Our main ingredient will be the following approximate max-plus convolution introduced by Chan [Cha18] (the proof is very similar to that of Lemma 3.27, which we give in the next section).

Lemma 3.22 ([Cha18, Item (i) from Lemma 1]). *Let $A[0\,..\,n], B[0\,..\,n]$ be two monotone non-decreasing sequences with t_1 and t_2 steps, respectively. Let $p_{\max}, p_{\min}$ be the maximum and minimum non-zero values in A, B. Then, for any $\varepsilon \in (0, 1)$ a monotone non-decreasing sequence C with $O(1/\varepsilon \cdot \log(p_{\max}/p_{\min}))$ steps which gives a (pointwise) $1 + O(\varepsilon)$-approximation of* MaxConv(A, B) *can be computed in time*

$$O((t_1 + t_2 + T(1/\varepsilon)) \cdot \log(p_{\max}/p_{\min}))$$

where $T(n)$ is the time needed to compute the min-plus convolution of two length-n sequences.

Now we are ready to describe the main routine. Since after the preprocessing we have that $w_{\min} > \varepsilon W$, any feasible solution contains at most $W/w_{\min} < 1/\varepsilon$ items. Thus, to approximate the optimal solution in $\mathcal{I}'$, we apply Lemma 3.17 for $O(\log 1/\varepsilon)$ iterations, but replace the max-plus convolutions with the approximate ones from Lemma 3.22. The pseudocode is written in Algorithm 7.

Lemma 3.23. *Let $(\mathcal{I}, W)$ be an instance of Unbounded Knapsack on n items with $p_{\min} > \varepsilon \mathrm{OPT}$ and $w_{\min} > \varepsilon W$. Then, on input $(\mathcal{I}, W)$ Algorithm 7 computes a pointwise $(1+O(\varepsilon))$-approximation of $\mathcal{P}_{\mathcal{I}}[0\,..\,W]$ in time $\widetilde{O}(n + T(1/\varepsilon))$, where $T(n)$ is the time to compute Bounded Monotone MaxPlus Conv on length-n sequences.*

Proof. First we argue about correctness. We claim that if the max-plus convolutions of Line 3 were *exact*, then for each i we have that $S_i = \mathcal{P}_i[0\,..\,W]$. For the base case $i = 0$, this holds by definition. For further iterations, this holds inductively by Lemma 3.17.

Algorithm 7 FPTAS($\mathcal{I}, W$): Given an instance $(\mathcal{I}, W)$ of Unbounded Knapsack with $w_{\min} > \varepsilon W$, the algorithm returns a $(1 + O(\varepsilon))$-approximation of $\mathcal{P}_{\mathcal{I}}[0\,..\,W]$

```
1  Initialize $S_0 := \mathcal{P}_0[0\,..\,W]$, stored implicitly
2  for $i = 1, \dots, \lceil \log 1/\varepsilon \rceil$ do
3      Approximate $S_i := (\textsc{MaxConv}(S_{i-1}, S_{i-1}))[0\,..\,W]$
       using Lemma 3.22 with error parameter $\varepsilon' := \varepsilon/\log(1/\varepsilon)$
4  return $S_{\lceil \log 1/\varepsilon \rceil}[0\,..\,W]$
```

Since we replaced the exact computations with approximate ones with error parameter $\varepsilon' = \varepsilon/\log(1/\varepsilon)$, and since there are $O(\log 1/\varepsilon)$ iterations where the error accumulates multiplicatively, we conclude that $S_{\lceil \log 1/\varepsilon \rceil}[0\,..\,W]$ is a pointwise $(1 + \varepsilon')^{O(\log(1/\varepsilon))} = 1 + O(\varepsilon' \log(1/\varepsilon)) = (1 + O(\varepsilon))$-approximation of $\mathcal{P}_{\lceil \log 1/\varepsilon \rceil}[0\,..\,W]$. Finally, note that since we assume that $w_{\min} > \varepsilon W$, any feasible solution consists of at most $W/w_{\min} < 1/\varepsilon$ items. This implies that $\mathcal{P}_{\lceil \log 1/\varepsilon \rceil}[0\,..\,W] = \mathcal{P}[0\,..\,W]$.

Now we argue about the running time. The initialization in Line 1 takes time $O(n)$, since by definition $\mathcal{P}_0[0\,..\,W]$ has n steps corresponding to the n solutions containing one item. Next, we look at the for-loop in Line 2. For each iteration, we use Lemma 3.22 to approximate the convolutions. Note that each application involves sequences with $O(n + 1/\varepsilon \log(p^*/p_{\min}))$ steps by the guarantee of Lemma 3.22 (the n term arising from the steps of S_0) where p^* is the maximum entry in any of the involved sequences. Since each entry in any of the sequences corresponds to a feasible solution for the instance, we have that $p^* \leq \mathrm{OPT}$. Moreover, by assumption we have that $p_{\min} \geq \varepsilon\mathrm{OPT}$, and hence $p^*/p_{\min} \leq O(1/\varepsilon)$. Therefore, each iteration takes time $\widetilde{O}(n + 1/\varepsilon + T(1/\varepsilon))$, and the overall time is bounded by

$$\sum_{i=1}^{\lceil \log \frac{1}{\varepsilon} \rceil} \widetilde{O}\left(n + \tfrac{1}{\varepsilon} + T(1/\varepsilon)\right) = \widetilde{O}(n + T(1/\varepsilon)).$$

Note that in this proof we are using the niceness Assumption 3.6 that $T(\widetilde{O}(n)) = \widetilde{O}(T(n))$. □

We put the pieces together to conclude the proof of the main theorem of this section.

Proof of Theorem 3.18. Given an instance of $(\mathcal{I}, W)$ of Unbounded Knapsack, we first run the preprocessing from Algorithm 6 and obtain a new instance $(\mathcal{I}', W)$. Then, we run Algorithm 7 on $(\mathcal{I}', W)$ and obtain a sequence $S[0\,..\,W]$. We claim that $S[W]$ gives the desired $(1 + O(\varepsilon))$-approximation to OPT in the desired running time.

To see the running time, note that after the preprocessing it holds that $p_{\min} > 2\varepsilon P_0 \geq \varepsilon\mathrm{OPT}$ and $w_{\min} > \varepsilon W$. Hence, we can apply Lemma 3.23 and conclude that the procedure runs in the desired running time. To argue about correctness, note that by Lemma 3.20 it holds that $\mathcal{P}_{\mathcal{I}'}[W] \geq (1 - 8\varepsilon)\mathrm{OPT}$. Combining this with the fact that $S[0\,..\,W]$

$(1 + O(\varepsilon))$-approximates $\mathcal{P}_{\mathcal{I}'}[0\,..\,W]$ due to Lemma 3.23, we conclude that

$$S[W] \geq \frac{\mathcal{P}_{\mathcal{I}'}[W]}{1 + O(\varepsilon)} \geq \frac{(1 - 8\varepsilon)\mathrm{OPT}}{1 + O(\varepsilon)} \geq \frac{\mathrm{OPT}}{1 + O(\varepsilon)},$$

as desired. □

3.4.3 Faster Weak FPTAS

In this section we prove the following theorem.

Theorem 3.24. *If Bounded Monotone MaxPlus Conv can be solved in time $T(n)$, then Unbounded Knapsack has a weak approximation scheme in time $\widetilde{O}(n + T(1/\varepsilon))$.*

Note that Main Theorem 1.6 follows as an immediate corollary by plugging in Chi, Duan, Xie and Zhang's algorithm (Corollary 3.3).

The Algorithm

Our approach is similar to the FPTAS from the previous section. The main difference is that since now we can overshoot the weight constraint by an additive term of εW, we can afford to round the weights of the items. By doing so, we can adapt the algorithm from Lemma 3.22 to use an algorithm for Bounded Monotone MaxPlus Conv.

We start by specifying the notion of approximation for monotone sequences we will be working with. In the following definition, it is helpful to think of the indices as weights and the entries as profits.

Definition 3.25 (Weak Approximation of a Sequence)**.** *We say that a sequence $A[0\,..\,n']$* weakly *$(1+\varepsilon)$-approximates a sequence $B[0\,..\,n]$ if the following two properties hold:*

(i) ***Good approximation for every entry.*** *For every $j \in [n]$, there exists $j' \in [n']$ with $A[j'] \geq B[j]/(1+\varepsilon)$ and $j' \leq (1+\varepsilon)j$.*

(ii) ***No approximate entry is better than a true entry.*** *For every $j' \in [n']$, there exists $j \in [n]$ such that $A[j'] \leq B[j]$ and $j' \geq j$.*

The following lemma shows that our notion of weak approximations is transitive. In particular, it implies that a weak approximation of the max-plus convolution of two weakly approximated sequences gives a (slightly worse) weak approximation of the convolution of the original sequences.

Lemma 3.26. *If A weakly $(1+\varepsilon)$-approximates B and B weakly $(1+\varepsilon')$-approximates C, then A weakly $(1+\varepsilon)(1+\varepsilon')$-approximates C.*

Proof. First we check that A and C satisfy property (i) of Definition 3.25. Fix an entry $C[k]$. Since B weakly $(1+\varepsilon')$-approximates C, by property (i) there exists an index j such that $B[j] \geq C[k]/(1+\varepsilon')$ and $j \leq (1+\varepsilon')k$. Thus, since A weakly $(1+\varepsilon)$-approximates B we apply property (i) once more to conclude that there is an index i such that $A[i] \geq B[j]/(1+\varepsilon) \geq \frac{C[k]}{(1+\varepsilon)(1+\varepsilon')}$ and $i \leq (1+\varepsilon)j \leq (1+\varepsilon)(1+\varepsilon')k$.

We can similarly check that A and C satisfy property (ii). Fix an entry $A[i]$ and apply property (ii) to obtain an index j such that $A[i] \leq B[j]$ and $i \geq j$. Applying property (ii) once more to j yields an index k such that $B[j] \leq C[k]$ and $j \geq k$. Combining, we conclude that $A[i] \leq C[k]$ and $i \geq k$, as desired. ◻

We extend Lemma 3.22 to give a weak approximation of the max-plus convolution of two sequences. Given two pairs of integers (w,p), (w',p'), we say that (w,p) *dominates* (w',p') if $w \leq w'$ and $p > p'$. Given a list of pairs $D = [(w_1,p_1),\dots,(w_m,p_m)]$, we can remove all dominated pairs from D in near-linear time for example by sorting.

Lemma 3.27. *Let $A[0\,..\,n], B[0\,..\,n]$ be two monotone non-decreasing sequences with t_1 and t_2 steps, respectively. Let $p_{\max}, p_{\min}$ be the maximum and minimum non-zero values in A, B and let $w_{\min} = \min\{\, i \mid A[i] \neq 0 \text{ or } B[i] \neq 0 \,\}$. Then, for any $\varepsilon \in (0,1)$ a monotone non-decreasing sequence C which is a weak $1+O(\varepsilon)$-approximation of $(\textsc{MaxConv}(A,B))[0\,..\,2n]$ and consists of $O(1/\varepsilon \cdot \log(p_{\max}/p_{\min}) \cdot \log(N/w_{\min}))$ steps can be computed in time*

$$O((t_1 + t_2 + T(1/\varepsilon)) \cdot \log(p_{\max}/p_{\min}) \log(N/w_{\min}))$$

where $T(n)$ is the time needed to compute Bounded Monotone MaxPlus Conv on sequences of length n.

Proof. Let $C := \textsc{MaxConv}(A,B)$. For this proof it will be convenient to work directly with the steps of the monotone sequences A, B and C. So suppose we are given A as a list of steps $A = [(w_1,p_1),\dots,(w_{t_1},p_{t_1})]$ where each step (w,p) represents an entry $p = A[w]$ at which A changes, i.e., $A[w-1] < A[w]$ (or $w = 0$). Similarly, B and C are represented as lists of steps.

Fix integers $p^*, w^* > 0$ and define a sequence of steps A' by keeping every step $(w,p) \in A$ with $w \leq w^*$ and $p \leq p^*$ and rounding it to $(\lceil w/(\varepsilon w^*)\rceil, \lfloor p/(\varepsilon p^*)\rfloor)$. Let B' be defined analogously. Compute $C' := \textsc{MaxConv}(A',B')$, and scale each step $(w',p') \in C'$ back, i.e., set $w := w' \cdot (\varepsilon w^*)$ and $p := p \cdot (\varepsilon p^*)$, obtaining a sequence of steps C''.

▷ Claim 3.28. The following holds:

(a) For every step $(w,p) \in C$ for which $w^*/2 \leq w \leq w^*$ and $p^*/2 \leq p \leq p^*$, there is a step $(w'',p'') \in C''$ such that $w'' \leq (1+4\varepsilon)w$ and $p'' \geq p/(1+8\varepsilon)$.

(b) For every step $(w'',p'') \in C''$ there is a step $(w,p) \in C$ such that $w'' \geq w$ and $p'' \leq p$.

Proof. We first prove Item (a). Fix one step $(w, p) \in C$ and let $(w_a, p_a) \in A$, $(w_b, p_b) \in B$ be the corresponding steps such that $w = w_a + w_b$ and $p = p_a + p_b$. Note that (w_a, p_a) has a corresponding rounded step $(w'_a, p'_a) \in A'$ since $w \le w^*$ and $p \le p^*$ imply that (w_a, p_a) were included (after rounding) to A'. Similarly, (w_b, p_b) has a corresponding step $(w'_b, p'_b) \in B'$. Hence, $C'[w'_a + w'_b] \ge p'_a + p'_b$, and therefore there is a step $(w''_c, p''_c) \in C''$ such that

$$w''_c \le (w'_a + w'_b) \cdot \varepsilon w^* \le (w/(\varepsilon w^*) + 2) \cdot \varepsilon w^* \le (1 + 4\varepsilon) w,$$

where we used the definition of w'_a, w'_b in the second inequality, and $w^*/2 \le w$ in the last one. Similarly, we have that

$$p''_c \ge (p'_a + p'_b) \cdot \varepsilon p^* \ge (p/(\varepsilon p^*) - 2) \cdot \varepsilon p^* \ge (1 - 4\varepsilon) p \ge p/(1 + 8\varepsilon),$$

where in the last step we assumed $\varepsilon \le \frac{1}{8}$, which is without loss of generality. This proves Item (a).

Item (b) follows because each step $(w'', p'') \in C''$ is a (scaled) sum of steps $(w'_a, p'_a) \in A'$ and $(w'_b, p'_b) \in B'$ which by construction are rounded steps $(w_a, p_a) \in A$, $(w_b, p_b) \in B$. Thus,

$$\begin{aligned} w'' &= (w'_a + w'_b) \cdot \varepsilon w^* = (\lceil w_a/(\varepsilon w^*) \rceil + \lceil w_b/(\varepsilon w^*) \rceil) \cdot \varepsilon w^* \ge w_a + w_b, \\ p'' &= (p'_a + p'_b) \cdot \varepsilon p^* = (\lfloor p_a/(\varepsilon p^*) \rfloor + \lfloor p_b/(\varepsilon p^*) \rfloor) \cdot \varepsilon p^* \le p_a + p_b, \end{aligned}$$

which implies that there is a step $(w, p) \in C$ such that $w \le w_a + w_b \le w''$ and $p \ge p_a + p_b \ge p''$. ◁

Claim 3.28 implies that if we apply the described procedure for every $w_{\min} \le w^* \le 2N$ and $p_{\min} \le p^* \le p_{\max}$ which are powers of 2 and combine the results by keeping the set of non-dominated steps, we obtain a weak $(1 + O(\varepsilon))$-approximation of C. Indeed, Item (a) implies property (i) of Definition 3.25 and Item (b) implies property (ii). The overall procedure is summarized in Algorithm 8.

Now we analyze the running time. For each p^* and w^*, we spend $O(t_1 + t_2)$ time to construct A' and B' in Line 6 and Line 7. The key step is the computation of C' in Line 9. Note that the steps constructed in Line 6 for A' define a monotone sequence $A'[1\,..\,\lfloor 1/\varepsilon \rfloor]$ where $A'[i] := \max\{p' \mid (w', p') \in A', w' \le i\}$, and $B'[1\,..\,\lfloor 1/\varepsilon \rfloor]$ is defined similarly. Thus, A', B' are monotone non-decreasing sequences of length $O(1/\varepsilon)$, and due to the rounding their entries are bounded by $O(1/\varepsilon)$. Thus, we can compute C' in time $T(1/\varepsilon)$ and the overall running time of the algorithm is $O((t_1 + t_2 + T(1/\varepsilon)) \cdot \log(p_{\max}/p_{\min}) \log(N/w_{\min}))$. □

Now we are ready to give the main algorithm. The idea is the same as in the FPTAS in Algorithm 7, except that we replace the max-plus convolution computations, i.e., we use Lemma 3.27 instead of Lemma 3.22. The full pseudocode is in Algorithm 9.

For the analysis, we will use the following notation. Given a sequence $A[0\,..\,n]$, and an integer $i \ge 1$ we denote by $(A[0\,..\,n])^i$ the result of applying 2^i max-plus convolutions of A with itself, i.e., produced by the following process:

Algorithm 8 Given monotone non-decreasing sequences A, B represented as lists of steps, the algorithm returns the steps of a weak $(1 + O(\varepsilon))$-approximation of $\textsc{MaxConv}(A, B)$.

```
1  $S := [\,]$
2  for $i = 0, 1, 2, \dots, \lceil \log(p_{\max}/p_{\min}) \rceil$ do
3      $p^* := 2^i \cdot p_{\min}$
4      for $j = 0, 1, 2, \dots, \lceil \log(N/w_{\min}) \rceil$ do
5          $w^* := 2^j \cdot w_{\min}$
6          $A' := \left[(\lceil \frac{w}{\varepsilon \cdot w^*} \rceil, \lfloor \frac{p}{\varepsilon \cdot p^*} \rfloor) \mid (w, p) \in A,\ w \le w^*, p \le p^*\right]$
7          $B' := \left[(\lceil \frac{w}{\varepsilon \cdot w^*} \rceil, \lfloor \frac{p}{\varepsilon \cdot p^*} \rfloor) \mid (w, p) \in B,\ w \le w^*, p \le p^*\right]$
8          Remove dominated pairs from $A'$ and $B'$
9          Compute $C' := \textsc{MaxConv}(A', B')$ using Corollary 3.3
10         $C'' := [(w \cdot \varepsilon w^*, p \cdot \varepsilon p^*) \mid (w, p) \in C']$
11         Append $C''$ to $S$
12 Remove dominated pairs in $S$ and sort the remaining points
13 return $S$
```

```
1  $B := \textsc{MaxConv}(A, A)$
2  repeat $2^i - 2$ times
3      $B := \textsc{MaxConv}(A, B)$
4  $(A[0\,..\,n])^i := B$
```

If $i = 0$, we set $(A[0\,..\,n])^0 := A[0\,..\,n]$. Note that the length of the resulting sequence is $2^i \cdot n + 1$. Moreover, note that by definition, for $i \ge 1$ it holds that $(A[0\,..\,n])^i = \textsc{MaxConv}((A[0\,..\,n])^{i-1}, (A[0\,..\,n])^{i-1})$.

Algorithm 9 $\textsc{Weak-FPTAS}(\mathcal{I}, W)$: Given an instance $(\mathcal{I}, W)$ of Unbounded Knapsack, the algorithm returns a weak $(1 + O(\varepsilon))$-approximation of $(\mathcal{P}_{\mathcal{I},0}[0\,..\,W])^{\lceil \log 1/\varepsilon \rceil}$

```
1  Initialize $S_0 := \mathcal{P}_0[0\,..\,W]$, stored implicitly
2  for $i = 1, \dots, \lceil \log 1/\varepsilon \rceil$ do
3      Approximate $S_i := \textsc{MaxConv}(S_{i-1}, S_{i-1})$ using Lemma 3.27
           with error parameter $\varepsilon' := \varepsilon/\log(1/\varepsilon)$
4  return $S_{\lceil \log 1/\varepsilon \rceil}$
```

Lemma 3.29. *Let $(\mathcal{I}, W)$ be an instance of Unbounded Knapsack on $n = |\mathcal{I}|$ items, where $p_{\min} > \varepsilon\mathrm{OPT}$ and $w_{\min} > \varepsilon W$. Then, on input $(\mathcal{I}, W)$ Algorithm 9 computes a weak $(1 + O(\varepsilon))$-approximation of $(\mathcal{P}_0[0\,..\,W])^{\lceil \log 1/\varepsilon \rceil}$ in time $\widetilde{O}(n + T(1/\varepsilon))$, where $T(n)$ is the time needed to compute Bounded Monotone MaxPlus Conv on length-n sequences.*

Proof. First we argue correctness. We will show by induction that for every $i \ge 0$ it holds that S_i is a weak $(1 + \varepsilon')^i$-approximation of $(\mathcal{P}_0[0\,..\,W])^i$. For the base case $i = 0$, this holds by definition of S_0. For the inductive step, take $i > 0$ and assume the claim holds for $i-1$. By the computation in line 3 of Algorithm 9, S_i weakly $(1+\varepsilon')$-approximates $\textsc{MaxConv}(S_{i-1}, S_{i-1})$. And by the inductive hypothesis, S_{i-1} is a weak $(1+\varepsilon')^{i-1}$-approximation of $(\mathcal{P}_0[0\,..\,W])^{i-1}$. Therefore, by Lemma 3.26 it follows that S_i weakly $(1+\varepsilon')^i$-approximates

$$\textsc{MaxConv}((\mathcal{P}_0[0\,..\,W])^{i-1}, (\mathcal{P}_0[0\,..\,W])^{i-1}) = (\mathcal{P}_0[0\,..\,W])^i.$$

Since $\varepsilon' = \varepsilon/\log(1/\varepsilon)$, we conclude that $S_{\lceil \log 1/\varepsilon \rceil}$ is a weak $(1+\varepsilon')^i = (1+O(\varepsilon))$-approximation of $(\mathcal{P}_0[0\,..\,W])^{\lceil \log 1/\varepsilon \rceil}$, as desired.

Now we analyze the running time. The initialization in Line 1 takes time $O(n)$. Then, in each execution of Line 3 of the for-loop we apply Lemma 3.27 to approximate the max-plus convolutions. When $i = 1$ the input sequences have $O(n)$ steps, and we obtain a sequence with $O(1/\varepsilon \log(p_{\max}/p_{\min}) \log(W/w_{\min}))$ steps which approximates $\textsc{MaxConv}(\mathcal{P}_0[0\,..\,W], \mathcal{P}[0\,..\,W])$. Continuing inductively, in the i-th iteration we approximate the max-plus convolution of sequences with

$$O(n + 1/\varepsilon \log(2^i p_{\max}/p_{\min}) \log(2^i W/w_{\min}))$$

steps. Since $i \le O(\log 1/\varepsilon)$, and by assumption $p_{\min} > \varepsilon\mathrm{OPT}$ and $w_{\min} > \varepsilon W$ it follows that the number of steps of the input sequences in every iteration are bounded by $\widetilde{O}(n + 1/\varepsilon)$. Therefore, each iteration takes time $\widetilde{O}(n + T(1/\varepsilon))$. Thus, we can bound the overall running time by

$$\sum_{i=1}^{\lceil \log \frac{1}{\varepsilon} \rceil} \widetilde{O}\left(n + \tfrac{1}{\varepsilon} + T(1/\varepsilon)\right) = \widetilde{O}(n + T(1/\varepsilon)).$$

Note that here we are using the niceness Assumption 3.6 that $T(\widetilde{O}(n)) = \widetilde{O}(T(n))$. □

Now we can put things together to prove the main theorem.

Proof of Theorem 3.24. Given a instance of $(\mathcal{I}, W)$ of Unbounded Knapsack, we first preprocess it with Algorithm 6 and obtain a new instance $(\mathcal{I}', W)$. Then, we run Algorithm 9 on $(\mathcal{I}', W)$ and obtain as output a sequence $S := S_{\lceil \log 1/\varepsilon \rceil}$. We claim that $\tilde{p} := S[\ell]$ where $\ell := (1+O(\varepsilon))W$ (for a sufficiently large hidden constant given by the guarantee of Lemma 3.29) is the profit of a feasible solution with profit $\tilde{p} \ge \mathrm{OPT}/(1+O(\varepsilon))$ and weight at most $(1+O(\varepsilon))W$, and that we can compute it in the desired running time.

To see the claim, first note that after the preprocessing, Lemma 3.20 guarantees that $p_{\min} > 2\varepsilon P_0 \ge \varepsilon\mathrm{OPT}$ and $w_{\min} > \varepsilon W$. Hence, we can apply Lemma 3.29 and conclude that we can compute S (and therefore $S[\ell]$) in the desired running time.

To argue about correctness, let x^* be the optimal solution for the original instance $(\mathcal{I}, W)$, i.e., $\mathcal{P}_{\mathcal{I}}[W] = p(x^*) = \mathrm{OPT}$. By Lemma 3.20, there is a solution $\tilde{x}$ from $\mathcal{I}'$ which satisfies $p(\tilde{x}) \ge (1-8\varepsilon)p(x^*) = (1-8\varepsilon)\mathrm{OPT}$ and $w(\tilde{x}) \le w(x^*) \le W$. Thus, it suffices to argue that $S[\ell]$ corresponds to a solution from $\mathcal{I}'$ which weakly-approximates $\tilde{x}$.

Since the maximum number of items in any feasible solution for $(\mathcal{I}', W)$ is at most $W/w_{\min} < 1/\varepsilon$, we can apply Lemma 3.17 inductively to obtain that

$$(\mathcal{P}_{\mathcal{I}',0}[0\mathinner{..}W])^{\lceil \log 1/\varepsilon \rceil}[0\mathinner{..}W] = \mathcal{P}_{\mathcal{I}'}[0\mathinner{..}W]. \tag{3.3}$$

Putting things together, observe that Lemma 3.29 guarantees that the output sequence S is a weak $(1+O(\varepsilon))$-approximation of $(\mathcal{P}_{\mathcal{I}',0}[0\mathinner{..}W])^{\lceil \log 1/\varepsilon \rceil}$. Therefore, by combining (3.3), property (i) of Definition 3.25 and monotonicity we conclude that $S[\ell] \geq \mathrm{OPT}/(1+O(\varepsilon))$. Moreover, property (ii) of Definition 3.25 guarantees that $S[\ell]$ indeed corresponds to a solution from $\mathcal{I}'$ of weight at most $(1+O(\varepsilon))W$, completing the proof. □

3.4.4 Solution Reconstruction

For both our strong and weak approximation schemes presented in Section 3.4, we only described how to obtain an approximation of the *value* of the optimal solution. We can also reconstruct the solution itself by computing the witnesses of each max-plus convolution (see Lemma 3.5), as we show in the following lemma.

Lemma 3.30. *A solution attaining the value given by Algorithm 9 can be found in time $\widetilde{O}(n + T(1/\varepsilon))$, where $T(n)$ is the time to compute Bounded Monotone MaxPlus Conv on length-n sequences.*

Proof Sketch. We run Algorithm 9, but additionally compute witnesses for each convolution. More precisely, consider the sequence S_i obtained at each iteration i of Algorithm 9. Note that via Lemma 3.5 we can obtain the witness array for every step in S_i by computing the witnesses of each max-plus convolution computed inside Lemma 3.27, which only adds a polylogarithmic overhead to the running time. After doing this, for every step in S_i we can obtain the corresponding two steps in S_{i-1} that define it in constant time (by doing a lookup in the witness arrays). Thus, to reconstruct the optimal solution we proceed by obtaining the steps which define the solution $S_{\lceil \log 1/\varepsilon \rceil}[(1+O(\varepsilon))W]$, and recursively find the steps which define them in the previous level. Proceeding in this way, we eventually hit the entries of $\mathcal{P}_0$, which correspond to the items from $\mathcal{I}'$ which correspond to the solution found. The correctness of this procedure follows simply because we trace back the computation which led to the output value and store the items found in the first level. Since we can lookup the witnesses in constant time and at the i-th level of recursion we have 2^i subproblems, the running time is $O(\sum_{i \leq \log 1/\varepsilon} 2^i) = O(1/\varepsilon)$. □

With an analogous proof we can show the same for the strong FPTAS:

Lemma 3.31. *A solution attaining the value given by Algorithm 7 can be found in time $\widetilde{O}(n + T(1/\varepsilon))$, where $T(n)$ is the time to compute max-plus convolution on length-n sequences.*

3.5 Equivalence

We complement the algorithmic results of this chapter by showing reductions in the opposite direction: Following the same chain of reductions as in [Cyg+19; KPS17] but starting from bounded monotone instances of max-plus convolution, we reduce Bounded Monotone MaxPlus Conv to $O(n)$ instances of Unbounded Knapsack with $O(\sqrt{n})$ items each, where it holds that $w_{\max}, p_{\max}, W$ and OPT are all bounded by $O(\sqrt{n})$. Instantiating this reduction for the exact and approximate setting, we show the following theorem.

Theorem 3.32 (Equivalence). *For any problems A and B from the following list, if A can be solved in time $\widetilde{O}(n^{2-\delta})$ for some $\delta > 0$, then B can be solved in randomized time $\widetilde{O}(n^{2-\delta/2})$:*

1. *Bounded Monotone MaxPlus Conv on sequences of length n*
2. *Unbounded Knapsack on n items and $W, \mathrm{OPT} = O(n)$*
3. *Knapsack on n items and $W, \mathrm{OPT} = O(n)$*
4. *Weak $(1+\varepsilon)$-approximation for Unbounded Knapsack on n items and $\varepsilon = \Theta(1/n)$*

On the one hand, Main Theorem 1.4 solves Unbounded Knapsack in time $\widetilde{O}(n + (p_{\max} + w_{\max})^{1.5})$ by using Chi, Duan, Xie and Zhang's subquadratic Bounded Monotone MaxPlus Conv algorithm [Chi+22] (Corollary 3.3). On the other hand, Theorem 3.32 shows that any algorithm solving Unbounded Knapsack in time $\widetilde{O}(n + (p_{\max} + w_{\max})^{2-\delta})$ can be transformed into a subquadratic Bounded Monotone MaxPlus Conv algorithm. This shows that both our exact and approximation algorithms take essentially *the only possible route* to obtain subquadratic algorithms, by invoking a Bounded Monotone MaxPlus Conv algorithm.

Proof Overview As mentioned in Section 1.2, Cygan et al. [Cyg+19] and Künnemann et al. [KPS17] independently showed a reduction from max-plus convolution to Unbounded Knapsack. In Section 3.5.1 we show that following the same chain of reductions from max-plus convolution to Unbounded Knapsack but instead starting from Bounded Monotone MaxPlus Conv, with minor adaptations we can produce instances of Unbounded Knapsack with $W, \mathrm{OPT} = O(n)$.

Together with our exact algorithm for Unbounded Knapsack, which we can phrase as a reduction to Bounded Monotone MaxPlus Conv, we obtain an equivalence of Bounded Monotone MaxPlus Conv and Unbounded Knapsack with $W, \mathrm{OPT} = O(n)$ — if one of these problems can be solved in subquadratic time, then both can. Note that for Unbounded Knapsack with $W, \mathrm{OPT} = O(n)$ a weak $(1+\varepsilon)$-approximation for $\varepsilon = \Theta(1/n)$ already computes an exact optimal solution. This yields the reduction from Bounded Monotone MaxPlus Conv to the approximate version of Unbounded Knapsack. We similarly obtain a reduction to Knapsack with $W, \mathrm{OPT} = O(n)$. This yields our equivalences from Theorem 3.32.

3.5.1 Retracing known reductions

We follow the reduction from max-plus convolution to Unbounded Knapsack of [Cyg+19] and [KPS17], but starting from Bounded Monotone MaxPlus Conv instead of general instances. In particular, we closely follow the structure of Cygan et al.'s proof. Their reduction proceeds in three steps using the following intermediate problems. Note that we define the bounded monotone (BM) versions of their problems.

Problem 3.33 (BMMaxConv UpperBound)**.** *Given monotone non-decreasing sequences* $A[0\,..\,n], B[0\,..\,n], C[0\,..\,2n]$ *with entries in* $[O(n)]$*. The task is to decide whether for all* $k \in [2n]$ *we have* $C[k] \geq \max_{i+j=k} A[i] + B[j]$*.*

Problem 3.34 (BMSuperAdditivity Testing)**.** *Given a monotone non-decreasing sequence* $A[0\,..\,n]$ *with entries in* $[O(n)]$*. The task is to decide whether for all* $k \in [n]$ *we have* $A[k] \geq \max_{i+j=k} A[i] + A[j]$

The first step is to reduce Bounded Monotone MaxPlus Conv to its decision version BMMaxConv UpperBound. The technique used in this step traces back to a reduction from min-plus matrix product to negative weight triangle detection in graphs due to Vassilevska Williams and Williams [WW18]. Our proof is very similar to [Cyg+19, Theorem 5.5], but we need some extra care to ensure that the constructed instances have bounded entries.

Proposition 3.35. *Given an algorithm for BMMaxConv UpperBound in time* $T(n)$*, we can find a violated constraint* $C[i+j] < A[i] + B[j]$ *if it exists in time* $O(T(n) \cdot \log n)$*.*

Proof. Suppose A, B, C form a NO-instance of BMMaxConv UpperBound and let $k^* = i^* + j^*$ be the smallest index for which we have a violated constraint $C[k^*] < A[i^*] + B[j^*]$. Note that for any $k < k^*$ the prefixes $A[0\,..\,k], B[0\,..\,k], C[0\,..\,k]$ form a YES instance of BMMaxConv UpperBound. Thus, we can do binary search over prefixes to find k^*. □

Lemma 3.36 (Bounded Monotone MaxPlus Conv $\longrightarrow$ BMMaxConv UpperBound)**.** *If* BMMaxConv UpperBound *can be solved in time* $T(n)$*, then Bounded Monotone MaxPlus Conv can be solved in time* $\widetilde{O}(n \cdot T(\sqrt{n}))$*.*

Proof. Let $A[0\,..\,n], B[0\,..\,n]$ be an input instance of Bounded Monotone MaxPlus Conv. We will describe a procedure which given a sequence C of length $2n+1$, outputs for each $k \in [2n]$ whether $C[k] \geq \max_{i+j=k} A[i] + B[j]$. Given this procedure, we can determine all entries of MaxConv(A, B) by a simultaneous binary search using C in $O(\log n)$ calls. Since A, B are monotone non-decreasing, the sequence of guessed values C will remain monotone in all iterations.

We split the input sequences of Bounded Monotone MaxPlus Conv as follows. Let $\Delta = O(n)$ be the largest value in A and B. Given an interval $I \subseteq [n]$, we denote by A_I the contiguous subsequence of A indexed by I. Among the indices $i \in [n]$, we mark every multiple of $\lceil\sqrt{n}\rceil$. We also mark the smallest index i with $A[i] \geq j \cdot \lceil\sqrt{\Delta}\rceil$, for every integer $1 \leq j \leq \sqrt{\Delta}$. Then we split A at every marked index, obtaining subsequences

$A_{I_0}, \dots, A_{I_a}$ for $a \le n' := \lceil \sqrt{n} \rceil + \lceil \sqrt{\Delta} \rceil$. We analogously construct intervals $J_0, \dots, J_b$ with $b \le n'$ which partition B.

We denote by $C[0\,..\,2n]$ the sequence which we use to binary search the values of $\textsc{MaxConv}(A, B)$. Recall that our goal is to determine for each k whether $C[k] \ge \max_{i+j=k} A[i] + B[j]$. We will describe an iterative procedure which returns a binary array $M[0\,..\,2n]$ where $M[k] = 1$ if we have determined that $C[k] < \max_{i+j=k} A[i] + B[j]$, and $M[k] = 0$ otherwise.

Initialize $M[k] = 0$ for all $k \in [2n]$. Iterate over each $(x, y) \in [a] \times [b]$. We now describe how to check if $A[i] + B[j] \le C[i + j]$ holds for all $i \in I_x$ and $j \in I_y$, or otherwise find a violated constraint using the oracle for BMMaxConv UpperBound. Let $L := \min A_{I_x} + \min B_{J_y}$ and $U := \max A_{I_x} + \max B_{J_y}$. We proceed in the following two steps:

1. Identify all indices $k \in I_x + I_y$ for which $C[k] < L$. Since $A[i] + B[j] \ge L$ for every $i \in I_x, j \in J_y$, we conclude that $C[k] < \max_{i+j=k} A[i] + B[j]$, and thus set $M[k] = 1$ for every such index.

2. Add extra dummy entries $C[2n+1] := \infty$, and $M[2n+1] = 0$. Let $\operatorname{next}(k) := \min\{\, j \mid j \ge k, M[j] = 0 \,\}$. We construct a new sequence C' by setting for every index $k \in I_x + I_y$:
$$C'[k] := \min\{\, C[\operatorname{next}(k)], U + 1 \,\}.$$
The purpose of the dummy entries is to set $C'[k] := U+1$ if there is no $k \le j \le 2n$ for which $M[j] = 0$. Due to the monotonicity of C, it follows that C' is also monotone. Further, it holds that $L \le C'[k] \le U + 1$ for every entry k. Intuitively, the purpose of $\operatorname{next}(k)$ is to ignore the entries which already have been marked $M[k] = 1$ in step 1, or in previous iterations.

   Now we want to find a violating index $C'[i + j] < A_{I_x}[i] + B_{J_y}[j]$ if it exists, using the BMMaxConv UpperBound oracle. In order to do so, we shift the values of A_{I_x}, B_{J_y} and C' appropriately. More precisely, let $L_A := \min A_{I_x}$ and $L_B := \min B_{J_y}$ be the lowest numbers in A_{I_x}, B_{J_y} respectively. We subtract L_A from every element in A_{I_x}, L_B from every element in B_{J_y} and $L_A + L_B$ from every element in C'. This makes all entries bounded by $O(n') = O(\sqrt{n})$, and a violating index in the resulting instance is a violating index before shifting since we subtract the same quantity from both sides of the inequality $C'[i + j] < A_{I_x}[i] + B_{J_y}[j]$. Thus, we can use Proposition 3.35 to find a violating index $k = i + j$ if it exists, and if so, set $M[\operatorname{next}(k)] := 1$.

We repeat step 2 until we find no more violating indices (recomputing the sequence C' in each iteration). Then we repeat the process with the next pair (x, y).

We claim that in this way, we correctly compute the array of violated indices M. To see correctness, note that step 1 is trivially correct due to the lower bound on any pair of sums. For step 2, note that if we find a violating index $k = i + j$ and $\operatorname{next}(k) = k$, then setting $M[k] = 1$ is clearly correct. If $\operatorname{next}(k) \ne k$, it means that we had marked the index $M[k] = 1$ in a previous iteration or in step 1. Due to the monotonicity of A, B and C and the definition of $z := \text{next}(k)$, we have that

$$C[z] = C'[k] = C'[i+j] < A[i] + B[j] \leq \max_{i'+j'=k} A[i'] + B[j'] \leq \max_{i'+j'=z} A[i'] + B[j'],$$

so marking $M[z] = 1$ is correct. Conversely, for any index $k = i + j$ such that $C[k] < A[i] + B[j]$ at some iteration we will consider the subsequences where $i \in I_x$ and $j \in I_y$ holds. Since we repeat step 2 until no more violating indices are found, the index k will be marked during some iteration.

Finally, we analyze the running time of the reduction. We consider $O(n)$ pairs $x, y \in [a] \times [b]$. For each such pair, step 1 takes time $O(n') = O(\sqrt{n})$. We execute step 2 at least once for each pair (x, y), and in total once for every index that we mark as violated. Since every index k gets marked at most once, this amounts to $O(n)$ calls to Proposition 3.35, which results in $O(n \log n)$ calls to the oracle of BMMaxConv UpperBound. Each such call takes time $O(T(\sqrt{n}))$. Since we do a simultaneous binary search over C, the overall time of the reduction is $O(n \cdot T(\sqrt{n}) \cdot \log^2 n)$. □

The next step is to reduce BMMaxConv UpperBound to BMSuperAdditivity Testing. The proof is exactly the same as [Cyg+19, Theorem 5.4]; we include it for completeness.

Lemma 3.37 (BMMaxConv UpperBound $\longrightarrow$ BMSuperAdditivity Testing)**.** *If BMSuper-Additivity Testing can be solved in time $T(n)$, then BMMaxConv UpperBound can be solved in time $O(T(n))$.*

Proof. Let $\Delta = O(n)$ be the maximum number in the input sequences A, B, C of BM-MaxConv UpperBound. We construct an equivalent instance $E[0\,..\,4n+3]$ of BMSuperAdditivity Testing as follows: for each $i \in [n]$ set $E[i] := 0$, $E[n+1+i] := \Delta + A[i]$, $E[2n+2+i] := 4\Delta + B[i]$ and $E[3n+3+i] := 5\Delta + C[i]$. Note that $|E| = O(n)$ and all values are bounded by $O(\Delta) = O(n)$.

If there are indices $i, j \in [n]$ such that $A[i] + B[j] > C[i+j]$, then $E[n+1+i] + E[2n+2+j] > E[3n+3+i+j]$, so E is a NO instance. Otherwise, let $i \leq j$ with $i + j \leq 4n+3$ be any pair of indices. If $i \leq n$, then since $E[i] = 0$ we have $E[i] + E[j] \leq E[i+j]$. So assume $i > n$. If $j \in [n+1\,..\,2n+1]$ then $E[i] + E[j] \leq 4\Delta \leq E[i+j]$. Hence, since $i \leq j$ and $i + j \leq 4n + 3$ we have that $i \in [n+1\,..\,2n+1]$ and $j \in [2n+2\,..\,3n+2]$. For these ranges, the super-additivity of E corresponds exactly to the BMMaxConv UpperBound condition of A, B, C. □

Finally, we reduce BMSuperAdditivity Testing to Unbounded Knapsack. The idea is essentially the same as [Cyg+19, Theorem 5.3], but in their proof they construct instances of Unbounded Knapsack where the maximum profit could be as large as $p_{\max} = O(n^2)$; we improve this to $p_{\max} = O(n)$. (More precisely, the variable D is defined as $D := \sum_i A[i]$ in their proof, while we show that it is enough to set $D = O(A[n])$.)

Lemma 3.38 (BMSuperAdditivity Testing $\longrightarrow$ Unbounded Knapsack)**.** *There is a linear-time reduction from BMSuperAdditivity Testing on sequences of length n to an instance of Unbounded Knapsack on n items with $W, \mathrm{OPT} = O(n)$.*

Proof. Let $A[0\,..\,n]$ be an instance of BMSuperAdditivity Testing. We construct an equivalent instance of Unbounded Knapsack as follows. Set $D := 5 \cdot A[n]$ and $W := 2n+1$. For every $i \in [n]$ we create a *light item* $(A[i], i)$ and a *heavy item* $(D - A[i], W - i)$, where the first entry of each item is its profit and the second is its weight. We set the weight constraint to W.

Suppose $A[0\,..\,n]$ is a NO instance. Then, there exist $i, j \in [n]$ such that $A[i]+A[j] > A[i+j]$. Hence, $(A[i], i)$, $(A[j], j)$ and $(D-A[i+j], W-i-j)$ form a feasible solution for Unbounded Knapsack with weight W and total profit $A[i] + A[j] + D - A[i+j] \geq D+1$.

On the other hand, suppose we start from a YES instance. Consider any feasible solution to the Unbounded Knapsack instance. We make two observations. First, if the solution contains any two light items $(A[i], i)$ and $(A[j], j)$ with $i + j \leq n$, then since A is a YES instance, we can replace both items by $(A[i + j], i + j)$ and the profit is at least as good. The second observation is that any feasible solution can contain at most one heavy item, since every such item has weight at least $n + 1$ and the weight constraint is $W = 2n + 1$. Now, suppose that OPT contains one heavy item of the form $(D - A[k], W - k)$. By the first observation, the most profitable way of packing the remaining capacity is to include the item $(A[k], k)$. Thus, in this case, we have that the value of OPT is $D - A[k] + A[k] = D$. If OPT does not contain any heavy item, note that by the first observation it consists of at most 4 light items. Therefore, its value is at most $4A[n] < D = 5A[n]$. ◻

Putting together the previous lemmas, we obtain the following theorem.

Theorem 3.39. *If Unbounded Knapsack on instances with n items with $W, \mathrm{OPT} = O(n)$ can be solved in time $T(n)$, then Bounded Monotone MaxPlus Conv on sequences of length n can be solved in time $\widetilde{O}(n \cdot T(\sqrt{n}))$.*

We will also need the following reduction from Unbounded Knapsack to Knapsack from [Cyg+19, Theorem 5.1]; we include the proof for completeness.

Lemma 3.40 (Unbounded Knapsack → Knapsack)**.** *If Knapsack on instances with n items and $W, \mathrm{OPT} = O(n)$ can be solved in time $T(n)$, then Unbounded Knapsack on instances with n items and $W, \mathrm{OPT} = O(n)$ can be solved in time $\widetilde{O}(T(n))$.*

Proof. Let $\mathcal{I} = \{\, (p_i, w_i) \,\}_{i \in [n]}$ with capacity W be an instance of Unbounded Knapsack. We construct an equivalent instance of Knapsack with the item set

$$\mathcal{I}' := \{(2^j p_i, 2^j w_i) \mid (p_i, w_i) \in \mathcal{I},\ 0 \leq j \leq \log W\},$$

and the same capacity W. Let $x \in \mathbb{N}^n$ be a solution of the Unbounded Knapsack instance $\mathcal{I}$, where x_i is the multiplicity of item (p_i, w_i). We can construct an equivalent solution $x' \in \{\,0, 1\,\}^{|\mathcal{I}'|}$ of the Knapsack instance $\mathcal{I}'$ by expressing each $x_i \leq W$ in binary and adding to x' the items from $\mathcal{I}'$ corresponding to the non-zero coefficients. In this way, $w_{\mathcal{I}}(x) = w_{\mathcal{I}'}(x')$ and $p_{\mathcal{I}}(x) = p_{\mathcal{I}'}(x')$. It is easy to see that this mapping can be inverted, which establishes the equivalence between the instances.

Note that this reduction does not change W and OPT, and n increases to $n \log W = O(n \log n)$. Thus, for Unbounded Knapsack we obtain running time $O(T(n \log n)) = \widetilde{O}(T(n))$. Here we used the niceness Assumption 3.6 on $T(n)$. □

3.5.2 Consequences

In this section we combine the reductions from Section 3.5.1, as well as our algorithms from Section 3.4, Section 3.2, and Section 3.3, to prove Theorem 3.32.

We start by showing how a weak approximation for Unbounded Knapsack gives an exact algorithm for Bounded Monotone MaxPlus Conv.

Lemma 3.41 (Bounded Monotone MaxPlus Conv $\longrightarrow$ Approximate Unbounded Knapsack)**.** *If Unbounded Knapsack has a weak approximation scheme running in time $T(n, \varepsilon)$, then Bounded Monotone MaxPlus Conv can be solved in time $O(n \cdot T(\sqrt{n}, 1/\sqrt{n}))$.*

Proof. Note that a weak approximation scheme for Unbounded Knapsack allows us to solve instances *exactly* by setting $\varepsilon := \Theta(1/(W + \mathrm{OPT}))$. Since the reduction from Theorem 3.39 produces $O(n)$ instances of Unbounded Knapsack with $\sqrt{n}$ items and $W, \mathrm{OPT} = O(\sqrt{n})$, we obtain an algorithm for Bounded Monotone MaxPlus Conv by setting $\varepsilon = \Theta(1/\sqrt{n})$. The overall running time becomes $O(n \cdot T(\sqrt{n}, 1/\sqrt{n}))$. □

Finally, we put the pieces together to prove Theorem 3.32.

Proof of Theorem 3.32. In the following, we write $A \longrightarrow B$ to denote a reduction from problem A to problem B. For every reduction stated in the following list, we obtain the stronger guarantee that if B can be solved in time $\widetilde{O}(n^{2-\delta})$, then A can be solved in time $\widetilde{O}(n^{2-\delta})$, i.e. without any loss in the exponent:

- (2) $\longrightarrow$ (1): Follows from Theorem 3.7.
- (2) $\longrightarrow$ (3): Follows from Lemma 3.40.
- (3) $\longrightarrow$ (1): Follows from Theorem 3.15.
- (4) $\longrightarrow$ (1): Follows from Theorem 3.24.

For the remaining two reductions below, we obtain the weaker guarantee as stated in the theorem. Namely, if B can be solved in time $\widetilde{O}(n^{2-\delta})$, then A can be solved in time $\widetilde{O}(n^{2-\delta/2})$:

- (1) $\longrightarrow$ (2): Follows from Theorem 3.39.
- (1) $\longrightarrow$ (4): Follows from Lemma 3.41.

Finally, note that for any pair of problems A and B in the theorem statement, we can reduce A to B by chaining the reductions written in the previous two lists and use at most one reduction which reduces the saving to $\delta/2$. This guarantees that if B can be solved in time $\widetilde{O}(n^{2-\delta})$, then A can be solved in time $\widetilde{O}(n^{2-\delta/2})$, as desired. □

Part II

Sublinear-Time Edit Distance

4 Introduction to Sublinear Edit Distance

This part of the thesis is based on our publication [Bri+24]. I contributed an equal share of the work, and more than half of the write-up. The main result in [Bri+24] builds upon our earlier publication [Bri+22b], which appeared in Nick Fischer's thesis. Here, for the sake of completeness, we extend the presentation of [Bri+24] by giving more details on the pieces that are based on [Bri+24].

[Bri+22b]	Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. "Almost-optimal sublinear-time edit distance in the low distance regime." In: *STOC*. ACM, 2022, pp. 1102–1115. DOI: 10.1145/3519935.3519990.
[Bri+24]	Karl Bringmann, Alejandro Cassis, Nick Fischer, and Tomasz Kociumaka. "Faster Sublinear-Time Edit Distance." In: *SODA*. SIAM, 2024, pp. 3274–3301. DOI: 10.1137/1.9781611977912.117.

Comparing texts is an essential primitive in computer science, with many applications in document processing, natural language processing, computational biology and many more domains. One of the most popular and well-studied ways to quantify the (dis)similarity of two strings X and Y is the *edit distance* $\mathrm{ED}(X, Y)$ (also known as Levenshtein distance [Lev66]), which is defined as the minimum number of character insertions, deletions, and substitutions to transform X into Y. A famous textbook dynamic programming algorithm computes the edit distance of two length-n strings in time $O(n^2)$ [Vin68; NW70; WF74; Sel74], and, despite considerable effort, this running time could only be improved by log-factors [MP80; Gra16]. More than 50 years after the initial efforts, this quadratic barrier can be explained using the modern toolkit from fine-grained complexity theory. More precisely, it has been shown that there is no $O(n^{2-\delta})$-time algorithm for any $\delta > 0$ under the Orthogonal Vectors Hypothesis [BI18; ABW15; BK15; Abb+16], which, in turn, follows from the Strong Exponential Time Hypothesis [IP01; IPZ01]. At the time though, Landau and Vishkin [LV88], building upon earlier results of Ukkonen [Ukk85] and Myers [Mye86], gave an elegant algorithm bypassing this barrier: It computes the edit distance of two strings in time $O(n + k^2)$, where the running time depends on the actual edit distance $k = \mathrm{ED}(X, Y)$. Note that indeed, this running time is subquadratic for $k \le n^{0.99}$. There is little hope to optimize the $O(n + k^2)$ time (beyond lower-order factors): Even restricted to instances with

$k = \Theta(n^\kappa)$ for some $\kappa \in (\frac{1}{2}, 1]$, a hypothetical $O(n + k^{2-\delta})$-time algorithm would directly yield an $O(n^{2-\delta})$-time algorithm for arbitrary edit distances, violating the fine-grained lower bound. Moreover, for $\kappa \in [0, \frac{1}{2}]$, the dominating $O(n)$ term is necessary simply to read the input strings.

Even before the aforementioned hardness results for exact edit distance were known, an extensive and beautiful body of work focused on *approximating* the edit distance in subquadratic time. The first major milestones of this long line of research include various polynomial-factor approximations [LMS98; Bar+04; BES06], a subpolynomial $n^{o(1)}$-factor approximation in almost-linear time $n^{1+o(1)}$ by Andoni and Onak [AO12] (based on the Ostrovsky-Rabani embedding of edit distance into ℓ_1 [OR07]), and a polylogarithmic $(\log n)^{O(1/\varepsilon)}$-factor approximation in almost-linear time $O(n^{1+\varepsilon})$ by Andoni, Krauthgamer and Onak [AKO10]. This polylogarithmic approximation remained the state of the art for several years until, only recently, Chakraborty, Das, Goldenberg, Koucký and Saks [Cha+20] achieved a breakthrough. Inspired by a quantum algorithm due to Boroujeni, Ehsani, Ghodsi, Hajiaghayi and Seddighin [Bor+21], they gave the first *constant*-factor approximation in subquadratic time $\widetilde{O}(n^{12/7})$.[1] Their work was later extended in two incomparable directions: On the one hand, subsuming [BR20; KS20b], Andoni and Nosatzki [AN20] improved the running time and developed a constant-factor approximation time $O(n^{1+\varepsilon})$ for any $\varepsilon > 0$ (where the approximation factor is a function that depends only on ε). On the other hand, Goldenberg, Saha and Rubinstein [GRS20] obtained an improved approximation factor of $3 + \varepsilon$ (for any constant $\varepsilon > 0$) in truly subquadratic time $n^{1.6+o(1)}$ (subsuming in running time [And20; Cha+20], which achieve the same approximation factor).

These results form an impressive state of the art for edit distance approximations, and it might seem that a constant-factor approximation in almost-linear time is close to the best that could be hoped for. However, it is, in principle, possible to expect *sublinear-time* approximation algorithms. In this regime it is standard to study approximations in the guise of the (k, K)-*gap edit distance* problem, where one is given random access to the strings X and Y, and the goal is to distinguish whether their edit distance at most k or more than K. An algorithm for this gap problem naturally extends to an approximation algorithm with multiplicative error K/k (the "gap").

To showcase that sublinear-time is a reasonable goal, let's consider as an inspiring success story the case of Hamming distance. Recall that the Hamming distance of two length-n strings is the number of positions in which they differ. This can be trivially computed in time $O(n)$. Moreover, a simple algorithm[2] solves the $(k, O(k))$-gap problem in time $O(n/k)$. This running time is unconditionally optimal, since there is a folklore $\Omega(n/k)$-time lower bound.

1. For readability in this introduction, we write $\widetilde{O}(\cdot)$, $\widetilde{\Omega}(\cdot)$ and $\widetilde{\Theta}(\cdot)$ to hide polylogarithmic factors $(\log n)^{O(1)}$. We also write $\widehat{O}(\cdot)$, $\widehat{\Omega}(\cdot)$ and $\widehat{\Theta}(\cdot)$ to hide subpolynomial factors $n^{o(1)}$. We do not use this notation in the technical parts to avoid ambiguities.
2. To distinguish e.g. k vs $4k$ Hamming distance, sample $n/(3k)$ indices uniformly at random, and output "$\leq k$" if and only if no mismatch is found among the sampled positions. We spare the details to the reader.

Coming back to edit distance, the natural question is whether we can match the linear-time state of the art by sublinear-time algorithms. Specifically:

Question 1: Is $(k, k^{1+o(1)})$-gap edit distance in truly sublinear time?
What about $(k, O(k))$-gap edit distance?

Of course, for small k, we cannot expect any sublinear-time improvements (as already distinguishing the all-zeros string from a string with a single one unconditionally requires reading $\Omega(n)$ characters), so by "truly sublinear" we mean running in time $n^{1-\Omega(1)}$ for $k \geq n^{\Omega(1)}$.

Fueled by this driving question, a line of research developed progressively better sublinear-time algorithms; see also Table 4.1 for the following list of relevant previous work. The first result in this direction is a seminal paper due to Batu, Ergün, Kilian, Magen, Raskhodnikova, Rubinfeld and Sami [Bat+03] who solved the $(k, \Theta(n))$-gap edit distance problem in sublinear time $\widetilde{O}(k^2/n + \sqrt{k})$ (subject to the restriction that $k < n^{1-\varepsilon}$ for some constant $\varepsilon > 0$). The aforementioned Andoni–Onak $n^{o(1)}$-approximation algorithm can be applied to obtain an algorithm for the (k, K)-gap edit distance problem in time $\widehat{O}(n^2 k/K^2)$ (provided that K is polynomially larger than k) [AO12]. More recently, based on the Landau–Vishkin algorithm, Goldenberg, Krauthgamer and Saha [GKS19] and Kociumaka and Saha [KS20a] solved the $(k, \Theta(k^2))$-gap problem in time $\widetilde{O}(n/k + k^2)$. Their algorithm allows for various other trade-offs between gap and running time (see Table 4.1). In a combined effort [Gol+22], they later developed a different algorithm (inspired by [Bat+03; AO12]) that runs in $\widetilde{O}(n\sqrt{k}/K + nk^2/K^2) \subseteq \widetilde{O}(nk/K)$ time for $K \geq k^{1+\Omega(1)}$. This solution is *non-adaptive* (i.e., the queried positions in the string can be fixed in advance). Moreover, they provided a strong barrier and proved that any *non-adaptive* algorithm for the (k, K)-gap problem requires $\Omega(n\sqrt{k}/K)$ queries. In that sense, their algorithm is optimal for $K \geq k^{3/2}$.

4.1 Our Results

Concurrently to the work of Goldenberg, Kociumaka, Krauthgamer and Saha [Gol+22], we resolved Question 1 and developed the first truly sublinear algorithm with a *subpolynomial* gap [Bri+22b]. The algorithm is adaptive, as is necessary by the previously mentioned lower bound [Gol+22]. In fact, we obtained the following two theorems which attain subpolynomial and polylogarithmic gap, with a small tradeoff in the running time.

Theorem 4.1. *The $(k, k \cdot 2^{\Theta(\sqrt{\log k \log\log k})})$-gap edit distance problem is in time $O(n/k + k^{2+o(1)})$.*

Theorem 4.2. *For any $\varepsilon \in (0, 1)$, the $(k, k \cdot (\log k)^{O(1/\varepsilon)})$-gap edit distance problem is in time $O(n/k^{1-\varepsilon} + k^{4+o(1)})$.*

Table 4.1: A comparison of sublinear-time algorithms for the (k, K)-gap edit distance problem for different gap parameters k and K. All algorithms in this table are randomized and succeed with high probability.

Source	Running time	Assumptions
Batu, Ergün, Kilian, Magen, Raskhodnikova, Rubinfeld, Sami [Bat+03]	$\widetilde{O}(k^2/n + \sqrt{k})$	$(k = n^{1-\Omega(1)}$ and $K = \Omega(n))$
Andoni, Onak [AO12]	$\widehat{O}(n^2 k/K^2)$	$(K > k^{1+\Omega(1)})$
Goldenberg, Krauthgamer, Saha [GKS19]	$\widetilde{O}(nk/K + k^3)$	$(K > k^{1+\Omega(1)})$
Kociumaka, Saha [KS20a]	$\widetilde{O}((nk + \sqrt{nk^5})/K + k^2)$	$(K > k^{1+\Omega(1)})$
Brakensiek, Charikar, Rubinstein [BCR20]	$\widetilde{O}((n + k^2) \cdot k^{3/2}/K)$	$(K \geq k^{3/2})$
Goldenberg, Kociumaka, Krauthgamer, Saha [Gol+22]	$\widetilde{O}(n\sqrt{k}/K)$	$(K \geq k^{3/2})$
Goldenberg, Kociumaka, Krauthgamer, Saha [Gol+22]	$\widetilde{O}(nk^2/K^2)$	$(k^{3/2} > K > k^{1+\Omega(1)})$
Bringmann, Cassis, Fischer, Nakos [Bri+22b]	$\widehat{O}(n/k + k^4)$	$(K = \widehat{\Theta}(k))$
This work (Corollary 4.4)	$\widehat{O}(n/k + k^2)$	$(K = \widehat{\Theta}(k))$
This work (Corollary 4.6)	$\widehat{O}(n/K + \sqrt{nk} + k^2)$	$(K > k^{1+\Omega(1)})$

For simplicity, we will throughout refer to the algorithm yielding Theorems 4.1 and 4.2 as the *BCFN algorithm.* In summary, the BCFN algorithm provides a satisfying and, perhaps, surprising answer to the question of whether we can expect accurate approximations in truly sublinear time. Nevertheless, the range of parameters for which it becomes effective is quite limited. For instance, for $k \geq n^{\frac{1}{4}-o(1)}$ the BCFN algorithm is even outperformed by the classic Landau–Vishkin algorithm [LV88] that computes the edit distance *exactly* in time $O(n + k^2)$. This state of affairs naturally raises the equally important follow-up question:

Question 2: What is the time complexity of the $(k, k^{1+o(1)})$-gap edit distance problem?

The strongest known unconditional lower bound is $\widehat{\Omega}(n/k + \sqrt{n})$ [Bat+03; AN10], where the n/k term is necessary already for the gap *Hamming* distance problem. In light of this lower bound, the BCFN running time $\widehat{O}(n/k + \text{poly}(k))$ has the right format, except that it remains to optimize the additive $\text{poly}(k)$ term.

Our contribution is that we drastically reduce the $\text{poly}(k)$ term, thereby making significant progress towards our driving Question 2. Specifically, our main result is the following technical theorem that we will shortly instantiate for several interesting parameter settings:

Main Theorem 4.3. *Let $2 \leq \Delta \leq n$ be a parameter. Then, there is a randomized algorithm that solves the (k, K)-gap edit distance problem in time $O(n/K+kK)\cdot\Delta^3\cdot(\log n)^{O(\log_\Delta n)}$ and succeeds with constant probability, provided that $K/k \geq (\log n)^{c\cdot\log_\Delta(n)}$ for a sufficiently large constant $c > 0$.*

The main consequence of Main Theorem 4.3 is a significant improvement for the $(k, k^{1+o(1)})$-gap edit distance problem.

Corollary 4.4 (Subpolynomial Gap)**.** *The $(k, k \cdot 2^{\Theta(\sqrt{\log k \log\log k})})$-gap edit distance problem is in time $O(n/k + k^{2+o(1)})$.*

Corollary 4.4 constitutes a *quadratic* improvement over the BCFN algorithm [Bri+22b]. As a consequence, we extend the range of parameters for which we know truly sublinear-time algorithms for the $(k, k^{1+o(1)})$-gap problem to $n^\varepsilon \leq k \leq n^{\frac{1}{2}-\varepsilon}$. In a slightly narrower range of $n^\varepsilon \leq k \leq n^{\frac{1}{3}-\varepsilon}$, our algorithm matches the unconditional lower bound [Bat+03; AN10] and is therefore *almost-optimal*, up to subpolynomial factors in the gap and in the running time.

Another notable consequence is that our running time is never out-performed by the Landau-Vishkin algorithm. In other words, our algorithm can even be viewed as an alternative to the $O(n + k^2)$-time Landau–Vishkin algorithm: It runs in faster sublinear time $\widehat{O}(n/k + k^2)$ at the cost of returning an approximate result.

Similarly to the BCFN algorithm, we also build on the original framework of Andoni, Krauthgamer and Onak [AKO10]. As a consequence, our Main Theorem 4.3 allows for more precise approximations with *polylogarithmic* gap, at the mild cost of increasing the running time by a small polynomial factor:

Corollary 4.5 (Polylogarithmic Gap)**.** *For any constant $\varepsilon > 0$, the $(k, k\cdot(\log k)^{\Theta(1/\varepsilon)})$-gap edit distance problem is in time $O(n/k^{1-\varepsilon} + k^{2+\varepsilon})$.*

Finally, we also give new results in the regime where K is polynomially larger than k.

Corollary 4.6 (Polynomial Gap)**.** *Let k, K be such that $K > k^{1+\varepsilon}$ for some constant $\varepsilon > 0$. Then the (k, K)-gap edit distance problem is in time $\widehat{O}(n/K + \sqrt{nk} + k^2)$.*

This polynomial gap regime was not explicitly studied in our previous work [Bri+22b]. However, using the BCFN algorithm as a black-box, one can infer an algorithm for this task in time $\widehat{O}(n/K + n^{0.8} + k^4)$. Our Corollary 4.6 improves upon this in all parameter regimes.

4.2 Technical Overview

In this section, we give a high-level overview of our new ideas used to obtain Main Theorem 4.3. The starting point for our result is the BCFN algorithm. It is based on two main ingredients: (1) The Andoni–Krauthgamer–Onak framework providing an efficient recursion scheme, based on the so-called *precision sampling* technique, and (2) a *structure-versus-randomness* dichotomy that is used to bound the number of recursive subproblems by $\mathrm{poly}(k)$. Since the algorithm is quite complex, and since our improvements are only concerned with the second ingredient, we omit an extensive description of the BCFN algorithm here; we refer the interested reader to our technical Chapter 5 (or to the original paper [Bri+22b] for a thorough overview).[3] Instead, we focus on highlighting our new key concepts.

One of the central ideas behind the BCFN algorithm is the notion of *block periodicity*, formally defined as follows:

Definition 4.7 (Block Periodicity). *Let X be a string and $p \geq 1$ be an integer. The* p-block periodicity $\mathrm{BP}_p(X)$ *of X is the smallest integer L such that X can be partitioned into L substrings, $X = \bigodot_{\ell=1}^{L} X_\ell$, so that each substring X_ℓ is p-periodic, that is, the period of X_ℓ is at most p.*

The block periodicity is a natural measure of *structure* in a string: On the one hand, strings with minimal block periodicity ($\mathrm{BP}_p(X) = 1$) are simply *periodic*. On the other hand, strings with very large block periodicity are composed of many non-repetitive parts and often behave like *random* strings. A frequent phenomenon in string algorithms is that specialized techniques for these two extreme cases ultimately lead to algorithms for the entire spectrum.

An important stepping stone towards the general BCFN algorithm for the $(k, k^{1+o(1)})$-gap edit distance problem is the special case where X has bounded block periodicity $\mathrm{BP}_p(X) \leq B$ whereas Y can be arbitrary. Let us refer to this special case as the *block-periodic case*. In our setting, p and B are parameters subject to some technical conditions (such as $p, B \geq k$) that will not concern us here. The BCFN algorithm can be interpreted in the following way:

1. An efficient algorithm for the block-periodic case. Specifically, it follows from our previous work [Bri+22b] that the block-periodic case can be solved in time $\widehat{O}(n/k + Bpk)$.

2. A reduction from the general to the block-periodic case. This reduction picks as parameters $p, B = \widehat{O}(k)$ and requires an additional multiplicative overhead of k. In combination with the previous item, the running time thus becomes $\widehat{O}(n/k + k^4)$.

We remark that this interpretation is not immediate from the original paper [Bri+22b] and requires rearranging the algorithm up to some degree.[4] In this work, we achieve our results by separately improving both of these steps.

3. In particular, throughout this overview, we will ignore the term "n/k" in all running times as dealing with these terms requires precision sampling which is not our focus here. More precisely, while in this overview we implicitly pretend that in each subproblem we query the strings at $\widehat{O}(n/k + \mathrm{poly}(k))$ positions, we in fact query at $\widehat{O}(pn + \mathrm{poly}(k))$ positions where p is a precision parameter distributed with $\mathbb{E}(p) \leq \widehat{O}(1/k)$.

4. In particular, in [Bri+22b] we did not explicitly studied the block-periodic case as a subproblem, nor did we consider p and B as parameters on their own.

Improvement 1: Speed-Up for the Block-Periodic Case As our first major contribution, we achieve a factor-k speed-up for the block-periodic case; see the following simplified statement:

Lemma 4.8 (Simplified Version of Lemma 5.1)**.** *Let X be a string with block periodicity $\mathrm{BP}_p(X) \le B$. The $(k, k^{1+o(1)})$-gap edit distance problem for X and an arbitrary string Y is in time $\widehat{O}(n/k + Bp)$.*

Our strategy refines the approach by [Bri+22b] as follows. Intuitively, the algorithm keeps splitting the string X into smaller substrings $X_1, \dots, X_N$ until all of them are p-periodic. Since the block periodicity of X was initially bounded by B, we construct at most $N = \widehat{O}(B)$ many substrings in this way. Moreover, for almost all such substrings X_i, we can assume that they perfectly match their respective substrings Y_i up to some shift—if there are more than k exceptions, we can immediately infer that the edit distance between X and Y exceeds k and terminate the algorithm. So far, there is no difference to the original BCFN algorithm. Our improvement lies in the treatment of the periodic pieces (X_i, Y_i). Specifically, we show the following lemma.

Lemma 4.9 (Simplified Version of Lemma 5.12)**.** *Let X, Y be strings of lengths n and $n + k$, respectively. Given an integer p that is a period of both strings, we can compute $\widehat{O}(1)$-approximations of the edit distances $\mathrm{ED}(X, Y[i\,..\,n+i))$ (for $i \in [0\,..\,k]$) in time $\widehat{O}(p + k)$.*

In comparison, the analogous result in [Bri+22b] computes the edit distances exactly (capped with $O(k)$), but the running time increases by a multiplicative factor k. The reason why we have to compute the edit distance for *many shifts* is due to the intricate recursion scheme of the Andoni–Krauthgamer–Onak framework, which is applied throughout under the hood.

Lemma 4.9 is proven by combining two ideas: First, using structural insights on the edit distance of periodic strings, we can reduce the problem to strings of length $O(p)$ (see Lemma 5.21; here, we incur a factor-3 loss in the approximation ratio). Second, we apply in a black-box fashion a known algorithm that computes the edit distance between two strings for many shifts. The Andoni–Onak algorithm [AO12] is suitable for this task but only attains a subpolynomial approximation. To improve our approximation factor to polylogarithmic, we instead use the very recent *dynamic* edit distance approximation algorithm by Kociumaka, Mukherjee, and Saha [KMS23]; see Section 5.3.1 for more details.

In summary, these ideas are sufficient to improve the BCFN algorithm to run in time $\widehat{O}(n/k + k^3)$. The following insights further reduce the running time to $\widehat{O}(n/k + k^2)$.

Improvement 2: Block Periodicity Decomposition Our second and technically much more challenging contribution is to improve the reduction from the general case to the block-periodic case. Specifically, building on the $\widehat{O}(n/k + Bp)$-time algorithm for the block-periodic case, our goal is to develop an $\widehat{O}(n/k + k^2)$-time algorithm for the general $(k, k^{1+o(1)})$-gap edit distance problem.

Our reduction diverges completely from the approach of [Bri+22b]. It instead hinges on more structural insights related to the block periodicity, especially on the notion of *breaks*. For a string X, we say that a position i is a *p-break* if i is a multiple of p and $X[i \,.\,.\, i+3p)$ is not p-periodic (see Definition 6.1). We exploit breaks in two ways. First, we observe that the number of p-breaks approximates the p-block periodicity of a string within a constant factor (see Lemma 6.2). Since we can test if $X[i \,.\,.\, i+3p)$ is a p-break in time $O(p)$ [KJP77], this insight yields an efficient sublinear-time subroutine estimating the block periodicity of a string. In the BCFN algorithm, the block periodicity of a string was never explicitly computed, and this new "Block Periodicity Test" on its own already vastly streamlines the BCFN algorithm.

Second, we use breaks to solve the $(k, k^{1+o(1)})$-gap problem in a more direct way. Suppose that $\mathrm{ED}(X, Y) \leq k$ and fix an optimal edit distance alignment between X and Y. Let us sample indices i which are multiples of k at rate $\approx 1/k^2$ and identify all k-breaks in X among them. Then, with constant probability, none of the sampled breaks contain errors from the optimal alignment. In particular, we can infer how the optimal alignment matches each of the sampled breaks. If the alignment matches a break at position i, then the break must have an exact occurrence in Y at position $j \in [i - \lfloor k/2 \rfloor \,.\,.\, i + \lfloor k/2 \rfloor]$ (recall that we assume $|X| = |Y|$). Moreover, since the break is not k-periodic, it cannot have more than one such exact occurrence. For each sampled break, we use exact pattern matching [KJP77] to find the unique match in Y. This allows us to split the instance into substrings $X_1, \dots, X_1$ and $Y_1, \dots, Y_s$ such that $\sum_i \mathrm{ED}(X_i, Y_i) = \mathrm{ED}(X, Y)$. Moreover, by the aforementioned correspondence between breaks and block periodicity, with good probability the k-block periodicity of each substring X_i is bounded by $\widetilde{O}(k)$. This splitting procedure forms the heart of our algorithm (see Lemma 6.3).

This suggests the following algorithm. If the k-block periodicity of X is $\widehat{O}(k)$, then the instance falls into the block-periodic case, and it can be solved directly using Lemma 4.8 in time $\widehat{O}(n/k + Bp) = \widehat{O}(n/k + k^2)$. Otherwise, we apply the splitting routine to partition the strings into substrings $X_1, \dots, X_s$ and $Y_1, \dots, Y_s$ in such a way that $\mathrm{ED}(X, Y) = \sum_i \mathrm{ED}(X_i, Y_i)$ and the block periodicity of each piece X_i is bounded by $\mathrm{BP}_k(X_i) \leq \widetilde{O}(k)$. It remains to distinguish whether $\sum_i \mathrm{ED}(X_i, Y_i) \leq k$ or $\sum_i \mathrm{ED}(X_i, Y_i) > K$ (for $K = k^{1+o(1)}$).

Naively recursing on all subproblems is not efficient enough as there could be too many of them. Instead, a more careful approach is to *subsample* the subproblems and to recurse only on few of them. This is exactly the right task for the *precision sampling* technique, which has already proven to be an instrumental tool for approximate edit distance [AKO10; Bri+22b; Bri+22c; KMS23]. We remark that, in our algorithm in Chapter 6, we apply the technique in a more elementary way, similar to [IW05]. This variant incurs an $O(\log K)$-factor loss in the approximation, but we can still use it, even when aiming for a polylogarithmic gap, because our recursion is relatively shallow compared to [AKO10].

4.3 Open Problems

We leave some open problems interwoven with some reflections:

1. *Closing the gap between upper and lower bounds.*
   While our result makes considerable progress towards answering Question 2 above, it does not settle it. In particular, is there an algorithm for the $(k, k^{1+o(1)})$-gap edit distance problem in time $\widehat{O}(n/k + \sqrt{n})$? Alternatively, can we prove a stronger (possibly conditional) lower bound?

2. *Constant gap.*
   Perhaps the most interesting future direction is to improve the gap from subpolynomial to constant. More precisely, can the $(k, O(k))$-gap edit distance problem be solved in time $\widehat{O}(n/k + \mathrm{poly}(k))$? We believe that our approach is hopeless to answer this question since we build upon the Andoni–Krauthgamer–Onak framework [AKO10], which inherently incurs a polylogarithmic factor in the approximation.

   Most previous sublinear-time algorithms for edit distance (including ours) have more or less followed a recipe: take a known exact or approximate algorithm for edit distance, and try to implement a sublinear-time version of it (by using subsampling and further ideas). For instance, [GKS19; KS20a; BCR20] can all be seen as sublinear-time implementations of the Landau–Vishkin algorithm [LV88]; while our work [Bri+22b; Bri+24] can be seen as a sublinear-time implementation of the Andoni–Krauthgamer–Onak algorithm [AKO10]. Thus, a natural speculative approach to answer the aforementioned open problem is to try to obtain a sublinear-time implementation of the Andoni–Nosatzki algorithm [AN20]. Since the latter is very technically involved, a reasonable first step might be to simplify that result in itself. Indeed, we believe that simplifying Andoni–Nosatzki can prove fruitful not just for sublinear-time algorithms, but could lead to further developments for edit distance in other settings.[5]

3. *Simple(r) algorithms?*
   We acknowledge that unpacking the complete details of our work is a non-trivial task. The main reason is that we build upon the Andoni–Krauthgamer–Onak framework, which is in itself complicated.[6] This state of affairs begs the question: Is there a *simple* algorithm for the $(k, k^{1+o(1)})$-gap edit distance in time $\widehat{O}(n/k + \mathrm{poly}(k))$? Of course, what constitutes as simple is rather elusive. For us, a measurable bar would be an algorithm that does not use precision sampling (which even though after some insights can become "intuitive", it greatly complicates the technical details and

5. As an example: the simplified re-interpretation of the Andoni–Onak–Krauthgamer algorithm [AKO10] that we presented in [Bri+22b] already led to improvements in edit distance computation with preprocessing [Bri+22c] and in dynamic edit distance [KMS23].
6. As far as we are aware, we were the first authors that modified/extended their work. This happened more than 10 years after the publication of [AKO10].

obfuscates a high level description) and/or departs from the recursive scheme of the Andoni–Krauthgamer–Onak algorithm [AKO10] (more precisely, that does not use the *tree distance*, see Chapter 5 for details).

4.4 Outline

We structure the remainder of this part of the thesis as follows. In Section 4.5 we state some formal preliminaries. In Chapter 5 we present the improved algorithm for the block-periodic case (Improvement 1 from the overview). Finally, in Chapter 6 we present our main algorithm (Improvement 2 from the overview) and prove our Main Theorem 4.3 and its corollaries.

4.5 Preliminaries

For integers $i, j \in \mathbb{Z}$, we write $[i\,..\,j) := \{\, i, i+1, \dots, j-1 \,\}$ and $[i\,..\,j] := \{\, i, i+1, \dots, j \,\}$. The sets $(i\,..\,j]$ and $(i\,..\,j)$ are defined analogously. We set $\mathrm{poly}(n) = n^{O(1)}$ and $\mathrm{polylog}(n) = (\log n)^{O(1)}$.

Strings A string $X = X[0] \dots X[n-1] \in \Sigma^n$ is a sequence of $|X| = n$ symbols from an alphabet Σ. We usually denote strings by capital letters X, Y, Z. For integers i, j we denote by $X[i\,..\,j)$ the substring with indices in $[i\,..\,j)$. Sometimes we call $X[i\,..\,j)$ a *fragment* of X. If the indices are out of bounds, we set $X[i\,..\,j) = X[\max(0,i)\,..\,\min(j, |X|))$, and similarly for $X[i\,..\,j]$.

For a string X and an integer s, we denote by $X^{\circlearrowleft s}$ the cyclical rotation of X defined as $X^{\circlearrowleft s}[i] = Y[(i+s) \bmod |X|]$ for $i \in [0\,..\,|X|)$. We say that X is *primitive* if all the non-trivial rotations of X are distinct from itself. For a string Q, we denote by Q^* the infinite-length string obtained by repeating Q. We say that X is *periodic with period* Q if $X = Q^*[0\,..\,|X|)$. For an integer $q \geq 1$, we say that X is *q-periodic* if it is periodic with some period of length at most q. We refer to the smallest period length of X as $\mathrm{per}(X)$.

Hamming and Edit Distances Given two strings X, Y of the same length, we define their Hamming distance $\mathrm{HD}(X,Y) := |\{\, i \mid X[i] \neq Y[i] \,\}|$ as the number of indices in which they differ. For two strings X, Y (with possibly different lengths), we define their *edit distance* $\mathrm{ED}(X,Y)$ as the minimum number of insertions, deletions or substitutions necessary to transform X into Y. We refer to insertions, deletions and substitutions as *edits*.

Given strings X and Y, an *alignment* is a monotonically non-decreasing function $A : \{\, 0, \dots, |X| \,\} \mapsto \{\, 0, \dots, |Y| \,\}$ such that $A(0) = 0$ and $A(|X|) = |Y|$. We say that A is an *optimal alignment* if it satisfies

$$\mathrm{ED}(X,Y) = \sum_{i=0}^{|X|-1} \mathrm{ED}(X[i], Y[A(i)\,..\,A(i+1))).$$

Intuitively, alignments correspond to a sequence of edits to transform X into Y. More precisely, alignments correspond to paths in the DAG defined by the standard dynamic programming algorithm to compute the edit distance between X and Y. Thus, an optimal alignment corresponds to a path of minimum cost.

The following simple proposition will be useful later.

Proposition 4.10 (Alignments Have Small Stretch)**.** *Let X, Y be strings of equal length. If A is an optimal alignment between X and Y, then $|i - A(i)| \leq \frac{1}{2}\,\mathrm{ED}(X, Y)$ holds for all $0 \leq i \leq |X|$.*

Proof. Since A is an optimal alignment between X and Y, we can write the edit distance $\mathrm{ED}(X, Y)$ as

$$\mathrm{ED}(X, Y) = \mathrm{ED}(X[0\,..\,i), Y[0\,..\,A(i))) + \mathrm{ED}(X[i\,..\,|X|), Y[A(i)\,..\,|Y|)).$$

Both edit distances in the right-hand side are at least $|i - A(i)|$, which is the length difference of these strings, respectively. It follows that $\mathrm{ED}(X, Y) \geq 2 \cdot |i - A(i)|$, as claimed. □

We formally define the gap edit distance problem as follows.

Definition 4.11 (Gap Edit Distance)**.** *The $\textsc{GapED}(k, K)$ problem is to distinguish, given two strings X, Y, whether*

- $\mathrm{ED}(X, Y) \leq k$ *(return* Close *in this case), or*
- $\mathrm{ED}(X, Y) > K$ *(return* Far *in this case).*

We say that an algorithm correctly solves the $\textsc{GapED}(k, K)$ problem if it returns the correct answer with constant probability.

Machine Model We work under the standard word RAM model, where the words have size logarithmic in the input size of the problem. That is, given input strings X, Y of total length n over an alphabet Σ, we assume the words have size $w = \Theta(\log n + \log |\Sigma|)$.

5 Algorithm for Bounded Block Periodicity

The goal of this chapter is to prove the following technical lemma, which constitutes a crucial stepping stone to obtain Main Theorem 4.3 (which we prove in Chapter 6).

Lemma 5.1 (Faster Algorithm for Bounded Block Periodicity)**.** *There exists an algorithm* AlgSmallBP *that, given two strings X, Y of length at most n, and parameters $k, K, p, B, \Delta \in \mathbb{Z}_+$ such that (i) $\mathrm{BP}_p(X) \leq B$, (ii) $p, B \geq k$, (iii) $(16 \log K)^2 \leq \Delta \leq n$, and (iv) $K/k \geq (\log n)^{\beta \cdot \log_\Delta(n)}$ where $\beta > 0$ is a constant, solves the* GapED(k, K) *problem with probability at least 0.9 and runs in time*

$$O\left(\left(\frac{n}{K} \cdot \Delta + pB \cdot \Delta\right) \cdot (\log n)^{\alpha \cdot \log_\Delta(n)}\right)$$

for some constant $\alpha > 0$.

The proof of Lemma 5.1 closely follows the approach of our previous paper [Bri+22b], which will be referred throughout as the BCFN algorithm.

Organization The outline of this chapter is as follows. In Section 5.1 we introduce the general *tree distance framework* pioneered by Andoni, Krauthgamer and Onak [AKO10], which is the starting point for our algorithm. In Section 5.2.1 we give the necessary technology to efficiently approximate the tree distance. Finally, in Section 5.3 we present and analyze the complete algorithm.

5.1 Tree Distance Framework

We start by introducing the general setup of the Andoni-Onak-Krauthgamer algorithm. The starting point is devising a way to split the computation of the edit distance into independent subtasks. A natural approach to do so would be to divide the two strings into equally sized blocks, compute the edit distances of the smaller blocks recursively, and combine the results. The difficulty in doing this is that the edit distance depends on a *global alignment*, which determines how the blocks should align and therefore the subproblems are not independent (e.g. the optimal alignment of one block might affect the optimal alignment of the next block). However, this can be overcome by computing the edit distances of one block in one string with several shifts of its corresponding block in the other string, and carefully combining the results. This type of *hierarchical decomposition* appeared in previous algorithms for approximating edit distance [AO12; AKO10; Bat+03; OR07]. In particular, Andoni, Krauthgamer and Onak [AKO10] define a string similarity measure called the *tree distance*[1] which gives a good approximation of the edit distance and cleanly splits the computation into independent subproblems.

To define the tree distance, we first need an underlying *partition tree*.

Definition 5.2 (Partition Tree). *Let X and Y be length-n strings. A partition tree T for X and Y is a balanced ℓ-ary tree with n leaves numbered from 0 to $n-1$ (from left to right). Each node v in T is associated with a multiplicative accuracy $\alpha_v > 1$ and a rate $r_v \geq 0$.*

For each node v in T, we define the substring X_v as follows: If the subtree below v spans from the i-th to the j-th leaf, then we set $X_v = X[i\,..\,j)$. Similarly, for a shift $s \in \mathbb{Z}$, we set $Y_{v,s} = Y[i+s\,..\,j+s)$ (the substring of Y relevant at v for one specific shift s).

With this at hand, we define the shift-restricted tree distance:

Definition 5.3 (Shift-Restricted tree distance). *Let T be a partition tree for length-n strings X and Y, and let $L \geq 0$ be an integer. For every node v in T and every shift $s \in [-L\,..\,L]$, we define the L-restricted tree distance $\mathrm{TD}^L_{v,s}(X,Y)$ as follows:*

- *If v is a leaf, then $\mathrm{TD}^L_{v,s}(X,Y) = \mathrm{ED}(X_v, Y_{v,s})$.*
- *If v is a node with children $v_0, \dots, v_{\ell-1}$, then*

$$\mathrm{TD}^L_{v,s}(X,Y) = \sum_{i=0}^{\ell-1} \widetilde{\mathrm{TD}}^L_{v_i,s}(X,Y), \tag{5.1}$$

where

$$\widetilde{\mathrm{TD}}^L_{v_i,s}(X,Y) = \min_{s' \in [-L\,..\,L]} \left(\mathrm{TD}^L_{v_i,s'}(X,Y) + 2|s-s'| \right). \tag{5.2}$$

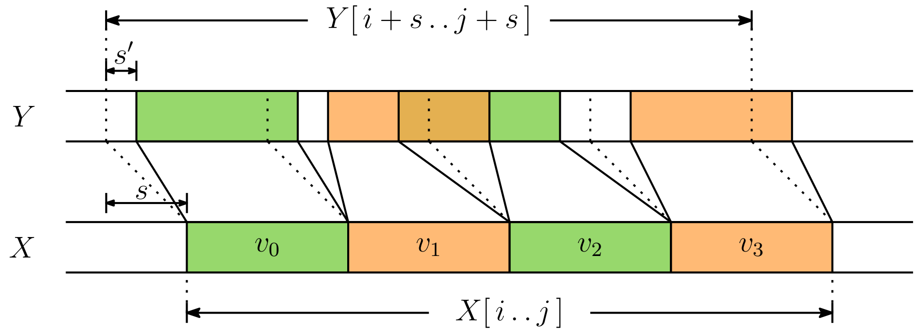
Since we restrict to shifts in $[-L\,..\,L]$, we define the substring of Y relevant at a node v as $Y_v := Y[i-L\,..\,j+L)$.

Figure 5.1 gives an illustration of this definition. We remark that compared to the tree distance definition in [AKO10; Bri+22b], in our Definition 5.3 we restrict the shifts to the set $[-L\,..\,L]$. The following lemma captures the relationship between the tree distance and edit distance.

Lemma 5.4 (Tree Distance is a good approximation of Edit Distance). *Let T be a partition tree for length-n strings X and Y, and let $L \geq 0$ be an integer. Suppose that T has maximum degree ℓ and height (the maximum distance from the root to a leaf) at most h. Then, the $\mathrm{TD}^L(X,Y)$ can be bounded as follows:*

1. Andoni et al. call the measure the $\mathcal{E}$-*distance*. However, in a talk by Robert Krauthgamer he recoined the name to *tree distance*. We decided to stick to this more descriptive name.

Figure 5.1: Illustrates the tree distance $\mathrm{TD}^L_{v,s}(X,Y)$ at a node v with interval $I_v = [i\mathinner{.\,.} j]$, shift s and children $v_0,\dots,v_3$. The dashed lines denote the shift given by s. The bold lines show the "local" shifts s' for each of the children (explicitly labeled for v_0). Note that some of these local shifts overlap.



- $\mathrm{TD}^L(X,Y) \ge \mathrm{ED}(X,Y)$;
- $\mathrm{TD}^L(X,Y) \le (2(\ell-1)h+1)\,\mathrm{ED}(X,Y)$ *provided that* $\mathrm{ED}(X,Y) \le L$.

Proof. For the lower bound, we inductively prove that

$$\mathrm{ED}(X_v, Y_{v,s}) \le \mathrm{TD}^L_{v,s}(X,Y) \qquad \text{and} \qquad \mathrm{ED}(X_v, Y_{v,s}) \le \widetilde{\mathrm{TD}}^L_{v,s}(X,Y)$$

hold for every node v and every shift $s \in [-L\mathinner{.\,.} L]$.

The first claim holds trivially if v is a leaf. Otherwise, we have $X_v = \bigodot_{i=0}^{\ell-1} X_{v_i}$ as well as $Y_{v,s} = \bigodot_{i=0}^{\ell-1} Y_{v_i,s}$. The subadditivity of edit distance and the inductive assumption imply

$$\mathrm{ED}(X_v, Y_{v,s}) \le \sum_{i=0}^{\ell-1} \mathrm{ED}(X_{v_i}, Y_{v_i,s}) \le \sum_{i=0}^{\ell-1} \widetilde{\mathrm{TD}}^L_{v_i,s}(X,Y) = \mathrm{TD}^L_{v,s}(X,Y).$$

To prove the second claim, observe that the following holds for every $s' \in [-L\mathinner{.\,.} L]$:

$$\mathrm{ED}(X_v, Y_{v,s}) \le \mathrm{ED}(X_v, Y_{v,s'}) + \mathrm{ED}(Y_{v,s'}, Y_{v,s}) \le \mathrm{TD}^L_{v,s'}(X,Y) + 2|s-s'|.$$

Consequently,

$$\mathrm{ED}(X_v, Y_{v,s}) \le \min_{s' \in [-L\mathinner{.\,.} L]} \Big(\mathrm{TD}^L_{v,s'}(X,Y) + 2|s-s'|\Big) = \widetilde{\mathrm{TD}}^L_{v,s}(X,Y).$$

For the upper bound, we prove the following claim for every node v and every shift $s \in [-L\mathinner{.\,.} L]$. Denote by h_v the height of the subtree rooted at v (that is, the maximum distance from v to a descendant of v). For two fragments $Y[p\mathinner{.\,.} q)$ and $Y[p'\mathinner{.\,.} q')$, denote $|Y[p\mathinner{.\,.} q) \triangle Y[p'\mathinner{.\,.} q')| = |p-p'| + |q-q'|$.

⊳ Claim 5.5. Let Y'_v be a fragment of Y such that $\mathrm{ED}(X_v, Y'_v) + |Y_{v,0} \triangle Y'_v| \le L$. Then,

$$\widetilde{\mathrm{TD}}^L_{v,s}(X, Y) \le (2(\ell - 1)h_v + 1)\,\mathrm{ED}(X_v, Y'_v) + |Y_{v,s} \triangle Y'_v|. \tag{5.3}$$

Moreover, if $|Y_{v,s} \triangle Y'_v| = \big||Y_{v,s}| - |Y'_v|\big|$, i.e., one of the fragments is contained in the other, then

$$\mathrm{TD}^L_{v,s}(X, Y) \le (2(\ell - 1)h_v + 1)\,\mathrm{ED}(X_v, Y'_v) + |Y_{v,s} \triangle Y'_v|. \tag{5.4}$$

Let us first prove (5.4). If v is a leaf, then $h_v = 0$ and simply

$$\mathrm{TD}^L_{v,s}(X, Y) = \mathrm{ED}(X_v, Y_{v,s}) \le \mathrm{ED}(X_v, Y'_v) + \mathrm{ED}(Y'_v, Y_{v,s}) \le \mathrm{ED}(X_v, Y'_v) + |Y_{v,s} \triangle Y'_v|.$$

Next, suppose that v has children v_i for $i \in [0\,..\,\ell)$. Decompose $X_v = \bigodot_{i=0}^{\ell-1} X[x_i\,..\,x_{i+1})$ so that $X_{v_i} = X[x_i\,..\,x_{i+1})$ and $Y_{v,s} = \bigodot_{i=0}^{\ell-1} Y[y_i\,..\,y_{i+1})$ so that $Y_{v_i,s} = Y[y_i\,..\,y_{i+1})$. Moreover, decompose $Y'_v = \bigodot_{i=0}^{\ell-1} Y[y'_i\,..\,y'_{i+1})$, denoting $Y'_{v_i} = Y[y'_i\,..\,y'_{i+1})$, so that $\mathrm{ED}(X_v, Y'_v) = \sum_{i=0}^{\ell-1} \mathrm{ED}(X_{v_i}, Y'_{v_i})$.

We shall inductively apply (5.3) for $X_{v_i,s}$ and Y'_{v_i}. For this, note that

$$\begin{aligned}
\mathrm{ED}(X_{v_i}, Y'_{v_i}) + |Y_{v_i,0} \triangle Y'_{v_i}| &= \mathrm{ED}(X_{v_i}, Y'_{v_i}) + |x_i - y'_i| + |x_{i+1} - y'_{i+1}| \\
&\le \mathrm{ED}(X_{v_i}, Y'_{v_i}) + |(x_i - x_0) - (y'_i - y'_0)| + |x_0 - y'_0| \\
&\qquad\qquad + |(x_\ell - x_{i+1}) - (y'_\ell - y'_{i+1})| + |x_\ell - y'_\ell| \\
&\le \mathrm{ED}(X[x_i\,..\,x_{i+1}), Y[y'_i\,..\,y'_{i+1})) + \mathrm{ED}(X[x_0\,..\,x_i), Y[y'_0\,..\,y'_i)) \\
&\qquad + |x_0 - y'_0| + \mathrm{ED}(X[x_{i+1}\,..\,x_\ell), Y[y'_{i+1}\,..\,y'_\ell)) + |x_\ell - y'_\ell| \\
&= \mathrm{ED}(X_v, Y'_v) + |Y_{v,0} \triangle Y'_v| \\
&\le L
\end{aligned}$$

Consequently, (5.3) yields

$$\widetilde{\mathrm{TD}}^L_{v_i,s}(X, Y) \le (2(\ell - 1)h_{v_i} + 1)\,\mathrm{ED}(X_{v_i}, Y'_{v_i}) + |y_i - y'_i| + |y_{i+1} - y'_{i+1}|.$$

The assumption $|Y_{v,s} \triangle Y'_v| = \big||Y_{v,s}| - |Y'_v|\big|$ translates to $|y_0 - y'_0| + |y_\ell - y'_\ell| = |(y_\ell - y_0) - (y'_\ell - y'_0)|$. Thus, the following holds for every $i \in [0\,..\,\ell]$:

$$\begin{aligned}
2|y_i - y'_i| &= |(y_i - y_0) - (y'_i - y'_0) + (y_0 - y'_0)| + |(y_i - y_\ell) - (y'_i - y'_\ell) + (y_\ell - y'_\ell)| \\
&\le |(y_i - y_0) - (y'_i - y'_0)| + |y_0 - y'_0| + |(y_i - y_\ell) - (y'_i - y'_\ell)| + |y_\ell - y'_\ell| \\
&= |(y_i - y_0) - (y'_i - y'_0)| + |(y_\ell - y_i) - (y'_\ell - y'_i)| + |(y_\ell - y_0) - (y'_\ell - y'_0)| \\
&= |(x_i - x_0) - (y'_i - y'_0)| + |(x_\ell - x_i) - (y'_\ell - y'_i)| + |(x_\ell - x_0) - (y'_\ell - y'_0)| \\
&\le \mathrm{ED}(X[x_0\,..\,x_i), Y[y'_0\,..\,y'_i)) + \mathrm{ED}(X[x_i\,..\,x_\ell), Y[y'_i\,..\,y'_\ell)) \\
&\qquad\qquad\qquad + \mathrm{ED}(X[x_0\,..\,x_\ell), Y[y'_0\,..\,y'_\ell)) \\
&= 2\,\mathrm{ED}(X_v, Y'_v).
\end{aligned}$$

Summing up over $i \in (0\,..\,\ell)$, due to $h_{v_i} \le h_v - 1$, we obtain

$$\begin{aligned}
\mathrm{TD}^L_{v,s}(X,Y) &= \sum_{i=0}^{\ell-1} \widetilde{\mathrm{TD}}^L_{v_i,s}(X,Y)\\
&\le \sum_{i=0}^{\ell-1} \big((2(\ell-1)h_{v_i}+1)\,\mathrm{ED}(X_{v_i},Y'_{v_i}) + |y_i - y'_i| + |y_{i+1} - y'_{i+1}|\big)\\
&\le (2(\ell-1)(h_v-1)+1)\,\mathrm{ED}(X_v,Y'_v)\\
&\qquad + 2(\ell-1)\,\mathrm{ED}(X_v,Y'_v) + |y_0 - y'_0| + |y_\ell - y'_\ell|\\
&= (2(\ell-1)h_v+1)\,\mathrm{ED}(X_v,Y'_v) + |Y_{v,s} \triangle Y'_v|.
\end{aligned}$$

It remains to argue that (5.3) follows from (5.4). Due to $\widetilde{\mathrm{TD}}^L_{v,s}(X,Y) \le \mathrm{TD}^L_{v,s}(X,Y)$, this is immediate if $|(y_\ell - y_0) - (y'_\ell - y'_0)| = \big||Y_{v,s}| - |Y'_v|\big| = |Y_{v,s} \triangle Y'_v| = |y_0 - y'_0| + |y_\ell - y'_\ell|$. Thus, we henceforth assume $\big||Y_{v,s}| - |Y'_v|\big| = \big||y_0 - y'_0| - |y_\ell - y'_\ell|\big|$. By symmetry, we may also assume without loss of generality that $|y_0 - y'_0| \le |y_\ell - y'_\ell|$, which implies $\big||Y_{v,s}| - |Y'_v|\big| = |y_\ell - y'_\ell| - |y_0 - y'_0|$.

In this case, we set $s' := y'_0 - x_0$. Note that $|s'| = |x_0 - y'_0| \le |Y_{v,0} \triangle Y'_v| \le L$, and thus $s' \in [-L\,..\,L]$. Moreover,

$$\begin{aligned}
|Y_{v,s'} \triangle Y'_v| &= |(x_0 + s') - y'_0| + |(x_\ell + s') - y'_\ell|\\
&= |(x_\ell + s' - x_0 - s') - (y'_\ell - y'_0)|\\
&= \big||Y_{v,s'}| - |Y'_v|\big|\\
&= \big||Y_{v,s}| - |Y'_v|\big|\\
&= |y_\ell - y'_\ell| - |y_0 - y'_0|\\
&= |y_0 - y'_0| + |y_\ell - y'_\ell| - 2|y_0 - y'_0|\\
&= |Y_{v,s} \triangle Y'_v| - 2|s - s'|.
\end{aligned}$$

In particular, $|Y_{v,s'} \triangle Y'_v| = \big||Y_{v,s'}| - |Y'_v|\big|$ lets us apply (5.4) for s', and thus

$$\begin{aligned}
\widetilde{\mathrm{TD}}^L_{v,s}(X,Y) &\le \mathrm{TD}^L_{v,s'}(X,Y) + 2|s-s'|\\
&\le (2(\ell-1)h_v+1)\,\mathrm{ED}(X_v,Y'_v) + |Y_{v,s'} \triangle Y'_v| + 2|s-s'|\\
&= (2(\ell-1)h_v+1)\,\mathrm{ED}(X_v,Y'_v) + |Y_{v,s} \triangle Y'_v|
\end{aligned}$$

holds as claimed. □

For our purpose, we will be using the tree distance with shifts restricted in $[-k\,..\,k]$, i.e. setting $L := k$ in Definition 5.3.

We are now ready to formally define the computational problem we are aiming to solve. Intuitively, we aim to compute the tree distance as defined in Definition 5.3. However, since we are aiming for sublinear time, we can only afford to approximate it. Recall that each node v in a partition tree T has an associated multiplicative accuracy $\alpha_v > 0$ and a rate $r_v \ge 0$. The task for each node in v is the following.

Definition 5.6 (Tree Distance Problem). *Let $\mu > 0$ be a parameter to be set. For every node v in the partition tree T, compute numbers $\eta_{v,-k}, \dots, \eta_{v,k}$ such that*

$$\frac{1}{\alpha_v}\,\mathrm{ED}(X_v, Y_{v,s}) - \frac{1}{r_v} \le \eta_{v,s} \le \alpha_v\,\mathrm{TD}^k_{v,s}(X, Y) + \frac{\mu}{r_v}. \tag{5.5}$$

5.2 Toolkit to Approximate the Tree Distance

In this section we introduce the tools needed to solve the tree distance problem. In Section 5.2.1 we introduce the technology that was used by Andoni, Krauthgamer and Onak to device their polylogarithmic near-linear time approximation for edit distance. Then, in Section 5.2.2 we introduce the ideas that were needed to obtain a sublinear-time implementation for the BCFN algorithm.

5.2.1 Andoni-Krauthgamer-Onak Tools

Suppose we are at some node v in the partition tree and we have already computed the values $\eta_{w,\cdot}$ for all its children w as per Definition 5.6. We want to use these approximations to compute the values $\eta_{v,\cdot}$ at v using the recursive definition of the tree distance given by Definition 5.3. If we do this naively, the additive error across the children adds up, which is prohibitely high. In order to control it, we use the following tool known as the *precision sampling lemma* as introduced by Andoni, Krauthgamer and Onak [AKO10].

Lemma 5.7 (Precision Sampling Lemma [And17]). *Fix parameters $\delta, \varepsilon > 0$. Let $\alpha \ge 1$ and $\beta \ge 0$. There is a distribution $\mathcal{D} = \mathcal{D}(\varepsilon, \delta)$ supported over $(0, 1]$ from which samples can be drawn in expected time $O(1)$, that satisfies the following:*

Accuracy *Let $a_1, \dots, a_\ell \ge 0$ be reals, and independently sample $u_1, \dots, u_\ell \sim \mathcal{D}$. There is an $O(\ell \cdot \varepsilon^{-2} \log(\delta^{-1}))$-time algorithm* Recover *satisfying for all $\widetilde{a}_1, \dots, \widetilde{a}_\ell$, with success probability at least $1 - \delta$:*

- *If $\widetilde{a}_i \ge \frac{1}{\alpha} \cdot a_i - \beta \cdot u_i$ for all i, then $\textsc{Recover}(\widetilde{a}_1, \dots, \widetilde{a}_\ell, u_1, \dots, u_\ell) \ge \frac{1}{(1+\varepsilon)\alpha} \cdot \sum_i a_i - \beta$.*
- *If $\widetilde{a}_i \le \alpha \cdot a_i + \beta \cdot u_i$ for all i, then $\textsc{Recover}(\widetilde{a}_1, \dots, \widetilde{a}_\ell, u_1, \dots, u_\ell) \le (1+\varepsilon)\alpha \sum_i a_i + \beta$.*

Efficiency *Sample $u \sim \mathcal{D}$. Then, for any $N \ge 1$ there is an event $\mathcal{E} = \mathcal{E}(u)$ happening with probability at least $1 - 1/N$, such that $\mathbb{E}_{u \sim \mathcal{D}}(\,1/u \mid \mathcal{E}\,) \le O(\varepsilon^{-2}\,\mathrm{polylog}(N, \delta^{-1}, \varepsilon^{-1}))$.*

The Precision Sampling Lemma was first shown in [AKO10] and later refined and simplified in [AKO11; And17]. We refer the reader to [Bri+22b, Appendix B] for a proof incorporating the simplifications hinted by Andoni [And17].

To efficiently evaluate the tree distance at some node v from the values of its children, we need the following efficient routine.

Lemma 5.8. *There is an $O(k)$-time algorithm for the following problem: Given integers $A_{-k}, \dots, A_k$, compute for all $s \in [-k \mathinner{.\,.} k]$:*

$$B_s = \min_{s' \in [-k \mathinner{.\,.} k]} A_{s'} + 2 \cdot |s - s'|.$$

Proof. The idea is to compute for each $s \in [-k \,..\, k]$ the auxiliary values

$$B_s^{\leq} = \min_{s' \in [-k \,..\, s]} A_{s'} + 2s - 2s',$$
$$B_s^{\geq} = \min_{s' \in [s \,..\, k]} A_{s'} - 2s + 2s',$$

as then returning $B_s = \min(B_s^{\leq}, B_s^{\geq})$ is correct.

We show how to compute $B_s^{\leq}$ for all s using dynamic programming; the values $B_s^{\geq}$ are symmetric. For the base case we set $B_{(-k)}^{\leq} = A_{-k}$, which is correct by definition. We then compute $B_s^{\leq}$ for all $s = -k+1, -k+2 \ldots, k$ as follows:

$$B_s^{\leq} = \min\left\{B_{s-1}^{\leq} + 2, A_s\right\}.$$

For the correctness, we distinguish two cases. Let $s' \leq s$ be the index which attains the minimum in the definition of $B_s^{\leq}$. On the one hand, if $s' < s$, then $B_s^{\leq} = B_{s-1}^{\leq} + 2$ and thus the first term in the minimum is correct. On the other hand, if $s' = s$, then the second term in the minimum is correct by definition. Finally, observe that we can compute $B_s^{\leq}$ for all $s \in [-k \,..\, k]$ by sweeping from left to right over all values in $[-k \,..\, k]$ exactly once, hence the running time is bounded by $O(k)$. ◻

Using precision sampling (Lemma 5.7) together with Lemma 5.8 to evaluate the tree distance essentially yields the original Andoni-Onak-Krauthgamer algorithm. See [Bri+22b, Appendix D] for our re-interpretation of their result.

5.2.2 Sublinear Tools: Property Testers

In order to obtain sublinear time, one needs a few more ideas and ingredients. The key idea used in the BCFN algorithm [Bri+22b] is to prune the computation of the tree distance once we reach nodes where we can easily infer the desired η values, instead of recursing to compute them. To this end, we introduced the following property testers.

The first is a simple (and folklore) tester to approximately test equality of strings.

Lemma 5.9 (Equality Test [Bri+22b, Lemma 23]). *Let X, Y be two strings of the same length, and let $r > 0$ be a sampling rate. There is an algorithm which returns one of the following two outputs:*

- *Close, in which case* $\mathrm{HD}(X, Y) \leq 1/r$.
- *Far, in which case $X \neq Y$.*

The algorithm runs in time $O(r|X|\log(\delta^{-1}))$ and is correct with probability at least $1 - \delta$.

Proof sketch. Sample $r|X|\log(1/\delta)$ positions $i \in [0 \,..\, |X|)$ uniformly at random and test if $X[i] = Y[i]$. If no error is found, return Close; otherwise, return Far. If $X = Y$, then the algorithm always returns Close. If $\mathrm{HD}(X, Y) > 1/r$, then the algorithm returns Far probability at least $1 - \delta$. ◻

Next, we designed an algorithm to test whether a string is close to being periodic.

Lemma 5.10 (p-Periodicity Test [Bri+22b, Lemma 17])**.** *Let X be a string, let $p \geq 1$ be an integer parameter and let $r > 0$ be a sampling rate. There is an algorithm which returns one of the following two outputs:*

- *Close(Q), where Q is a primitive string of length $\leq p$ with $\mathrm{HD}(X, Q^*[0\,..\,|X|)) \leq 1/r$.*
- *Far, in which case X is not p-periodic.*

The algorithm runs in time $O(r|X|\log(\delta^{-1}) + p)$ and is correct with probability $1 - \delta$.

Proof. We start analyzing the length-$2p$ prefix $Y = X[0\,..\,2p]$. In time $O(p)$ we can compute the smallest string Q with $|Q| \leq p$ such that $Y = Q^*[0\,..\,|Y|]$ by searching for the first match of Y in $Y \circ Y$, e.g. using the Knuth-Morris-Pratt pattern matching algorithm [KJP77]. If no such match exists, we can immediately report Far. So suppose that we find a period Q. It must be primitive (since it is the smallest such period) and it remains to test whether X globally follows the period. For this task we use the Equality Test (Lemma 5.9) with inputs X and Q^* (of course, we cannot write down the infinite string Q^*, but we provide oracle access to Q^* which is sufficient here). On the one hand, if X is indeed periodic with period Q, then the Equality Test reports Close. On the other hand, if X is $1/r$-far from any periodic string, then it particular $\mathrm{HD}(X, Q^*) > 1/r$ and therefore the Equality Test reports Far. The only randomized step is the Equality Test. We therefore set the error probability of the Equality Test to δ and achieve total running time $O(r|X|\log(\delta^{-1}) + K)$. □

We say that a node v in the partition tree is *matched* if there is a shift $s^* \in [-k\,..\,k]$ such that $X_v = Y_{v,s^*}$. The final and most intricate property tester checks whether a node is close to being matched, in the sense defined above.

Lemma 5.11 (Matching Test [Bri+22b, Lemma 18])**.** *Let X, Y be strings such that $|Y| = |X| + 2k$, and let $r > 0$ be a sampling rate. There is an algorithm which returns one of the following two outputs:*

- *Close(s^*), where $s^* \in [-k\,..\,k]$ satisfies $\mathrm{HD}(X, Y[k + s^*\,..\,|X| + k + s^*)) \leq 1/r$.*
- *Far, in which case there is no $s^* \in [-k\,..\,k]$ with $X = Y[k + s^*\,..\,|X| + k + s^*)$.*

The algorithm runs time $O(r|X|\log(\delta^{-1}) + k\log|X|)$ and is correct with probability $1 - \delta$.

Proof. For convenience, we write $Y_s = Y[K{+}s\,..\,|X|{+}K{+}s]$. Our goal is to obtain a single *candidate shift* s^* (that is, knowing s^* we can exclude all other shifts from consideration). Having obtained a candidate shift, we can use the Equality Test (Lemma 5.9 with parameters r and $\delta/3$) to verify whether we indeed have $X = Y_{s^*}$. In the positive case, Lemma 5.9 implies that $\mathrm{HD}(X, Y_{s^*}) \leq 1/r$, hence returning s^* is valid. The difficulty lies in obtaining the candidate shift. Our algorithm proceeds in three steps:

1. **Aligning the Prefixes:** We start by computing the set S consisting of all shifts s for which $X[0\,..\,2K] = Y_s[0\,..\,2K]$. One way to compute this set in linear time $O(K)$ is by using a pattern matching algorithm with pattern $X[0\,..\,2K]$ and text $Y[0\,..\,4K]$ (like the Knuth-Morris-Pratt algorithm [KJP77]). It is clear that S must contain any shift s for which globally $X = Y_s$. For that reason we can stop if $|S| = 0$ (in which case we return Far) or if $|S| = 1$ (in which case we test the unique candidate shift $s^* \in S$ and report accordingly).

2. **Testing for Periodicity:** After running the previous step we can assume that $|S| \geq 2$. Take any elements $s < s'$ from S; we have that $X[0\,..\,2K] = Y_s[0\,..\,2K] = Y_{s'}[0\,..\,2K]$. It follows that $X[0\,..\,2K - s' + s] = X[s' - s\,..\,2K]$, and thus by the periodicity lemma [FW65] we conclude that $X[0\,..\,2K]$ is periodic with period $P = X[0\,..\,s' - s]$, where $|P| \leq s' - s \leq 2K$. Obviously the same holds for $Y_s[0\,..\,2K]$ and $Y_{s'}[0\,..\,2K]$.

   We will now test whether X and Y_s are also globally periodic with this period P. To this end, we apply the Equality Test two times (each time with parameters $2r$ and $\delta/3$) to check whether $X = P^*[0\,..\,|X|]$ and $Y_s = P^*[0\,..\,|Y_s|]$. If both tests return Close, then Lemma 5.9 guarantees that $\mathrm{HD}(X, P^*[0\,..\,|X|]) \leq 1/(2r)$ and $\mathrm{HD}(Y_s, P^*[0\,..\,|Y_s|]) \leq 1/(2r)$ and hence, by the triangle inequality, $\mathrm{HD}(X, Y_s) \leq 1/r$. Note that we have witnessed a matching shift $s^* = s$.

3. **Aligning the Leading Mismatches:** Assuming that the previous step did not succeed, one of the Equality Tests returned Far(i_0) for some position $i_0 > 2K$ with $X[i_0] \neq P^*[i_0]$ or $Y_s[i_0] \neq P^*[i_0]$. Let us refer to these indices as *mismatches*. Moreover, we call a mismatch i a *leading mismatch* if the $2K$ positions to the left of i are not mismatches. We continue in two steps: First, we find a leading mismatch. Second, we turn this leading mismatch into a candidate shift.

   3a **Finding a Leading Mismatch:** To find a leading mismatch, we use the following binary search-style algorithm: Initialize $L \leftarrow 0$ and $R \leftarrow i_0$. We maintain the following two invariants: (i) All positions in $[L\,..\,L + 2K]$ are not mismatches, and (ii) R is a mismatch. Both properties are initially true. We will now iterate as follows: Let $M \leftarrow \lceil (L + R)/2 \rceil$ and test whether there is a mismatch $i \in [M\,..\,M + 2K]$. If there is such a mismatch i, we update $R \leftarrow i$. Otherwise, we update $L \leftarrow M$. It is easy to see that in both cases both invariants are maintained. Moreover, this procedure is guaranteed to make progress as long as $L + 4K < R$. If at some point $R \leq L + 4K$, then we can simply check all positions in $[L\,..\,R]$—one of these positions must be a leading mismatch i.

   3b **Finding a Candidate Shift:** Assume that the previous step succeeded in finding a leading mismatch i. Then we can produce a single candidate shift as follows: Assume without loss of generality that $X[i] \neq P^*[i]$, and let $i \leq j$ be the smallest position such that $Y_s[j] \neq P^*[j]$. Then $s^* = s + j - i$ is the only candidate shift (if it happens to fall into the range $\{-K, \dots, K\}$).

   Indeed, for any $s'' > s^*$ we can find a position where X and $Y_{s''}$ differ. To see this, we should assume that s'' respects the period (i.e., $P^* = P^*[K + s'' \mathinner{.\,.} \infty]$), since otherwise we find a mismatch in the length-$2K$ prefix. But then

$$\begin{aligned} Y_{s''}[j + s - s''] &= Y_s[j] && (5.6)\\ &\neq P^*[j] && (5.7)\\ &= P^*[j + s - s''] && (5.8)\\ &= X[j + s - s''], && (5.9) \end{aligned}$$

which proves that $X \neq Y_{s''}$ and thereby disqualifies s'' as a feasible shift. Here we used (5.6) the definition of Y_s, (5.7) the assumption that $Y_s[j] \neq P^*[j]$, (5.8) the fact that both s and s'' respect the period P and (5.9) the assumption that i was a leading mismatch which implies that X matches P^* at the position $j + s - s'' < i$.

A similar argument works for any shift $s'' < s^*$. In this case one can show that $X[i] \neq P^*[i] = Y_{s''}[i]$ which also disqualifies s'' as a candidate shift.

We finally bound the error probability and running time of this algorithm. We only use randomness when calling the Equality Test which runs at most three times. Since each time we set the error parameter to $\delta/3$, the total error probability is δ as claimed. The running time of the Equality Tests is bounded by $O(r|X|\log(\delta^{-1}))$ by Lemma 5.9. In addition, steps 1 and 2 take time $O(K)$. Step 3 iterates at most $\log|X|$ times and each iteration takes time $O(K)$. Thus, the total running time is $O(r|X|\log(\delta^{-1}) + K\log|X|)$. □

We remark that *after* the publication of [Bri+22b], we found out that Kociumaka and Saha had presented essentially the same algorithm as Lemma 5.11 in [KS20a, Proposition 3.4], but was used for a different purpose and phrased in a different way.

5.3 Algorithm

Now we are ready to present the algorithm yielding Lemma 5.1. The idea is to approximately evaluate the tree distance (Definition 5.3) over the partition tree. In order to obtain sublinear running time, we want to avoid recursing over the entire tree. For that end, we use the insight that whenever we encounter as a node v that is matched and p-periodic (which we can test using Lemmas 5.10 and 5.11), we can stop the recursive computation and approximate the values $\eta_{v,s}$ directly for all shifts $s \in [-k \mathinner{.\,.} k]$, as captured by Lemma 5.13. The rest of the algorithm combines the recursive results from the children using Lemma 5.7 and Lemma 5.8. Consult Algorithm 10 for the full pseudocode.

We start by proving Lemma 5.13. To achieve that, we provide an algorithm that can efficiently approximate the edit distance of strings that are periodic. More precisely, we leverage the following result whose proof is defered to Section 5.3.1.

Algorithm 10

Input: Strings X, Y, a node v in the partition tree T, integers $k, K, p \geq 0$ and a rate r_v
Output: $\eta_{v,s}$ for all shifts $s \in [-k\,..\,k]$

```
1  if $v$ is a leaf then
2      Compute and return $\eta_{v,s} = \mathrm{ED}(X_v, Y_{v,s})$ for all $s \in [-k\,..\,k]$
3  Run the Matching Test (Lemma 5.11) for $X_v, Y_v$ (with $r = 3r_v$ and $\delta = 0.01/n$)
4  Run the $p$-Periodicity Test (Lemma 5.10) for $Y_v$ (with $r = 3r_v$ and $\delta = 0.01/n$)
5  if the Matching Test returns $\textsc{Close}(s^*)$ then
6      if the Periodicity Test returns $\textsc{Close}(Q)$ then
7          Compute and return $\eta_{v,s}$ for all $s \in [-k\,..\,k]$ using Lemma 5.13
8  for each $i \in [0\,..\,\ell)$ do
9      Let $v_i$ be the $i$-th child of $v$ and sample $u_{v_i} \sim \mathcal{D}(\varepsilon := (2\log n)^{-1}, \delta := 0.01/(kn))$
10     Recursively compute $\eta_{v_i,s}$ with rate $r_v/u_{v_i}$ for all $s \in [-k\,..\,k]$
11     Compute $\widetilde{A}_{i,s} = \min_{s' \in [-k\,..\,k]} \eta_{v_i,s'} + 2 \cdot |s - s'|$ using Lemma 5.8
12 for each $s \in [-k\,..\,k]$ do
13     $\eta_{v,s} = \textsc{Recover}(\widetilde{A}_{0,s}, \ldots, \widetilde{A}_{\ell-1,s}, u_{v_0}, \ldots, u_{v_{\ell-1}})$
14 return $\eta_{v,s}$ for all $s \in [-k\,..\,k]$
```

Lemma 5.12. *Given a positive integer p, two strings P and T with period p and lengths $m \leq n$, respectively, and an integer $\Delta \in [2\,..\,p]$, one can compute, for all $i \in [0\,..\,n-m]$, multiplicative $(\log p)^{O(\log_\Delta p)}$-approximations of $\mathrm{ED}(P, T[i\,..\,i+m))$ in time $O(n-m) + p\Delta \cdot (\log p)^{O(\log_\Delta p)}$ correctly w.h.p.*

Lemma 5.13 (Fast Shifted ED for Periodic Strings)**.** *Let v be a node for which the Matching Test correctly returns $\textsc{Close}(s^*)$ and the Periodicity Test correctly returns $\textsc{Close}(Q)$ (i.e., Lines 5 to 6 succeed). Then, with high probability, given integers $p \geq |Q|$ and $\Delta \in [2\,..\,p]$, we can compute values $\eta_{v,s}$ for all $s \in [-k\,..\,k]$ such that*

$$\mathrm{ED}(X_v, Y_{v,s}) - \tfrac{1}{r_v} \leq \eta_{v,s} \leq (\log p)^{O(\log_\Delta(p))} \cdot \left(\mathrm{ED}(X_v, Y_{v,s}) + \tfrac{1}{r_v}\right),$$

in total time $O\left(p(\log p)^{O(\log_\Delta(p))}\Delta + k\right)$.

Proof. Denote $T = Q^*[0\,..\,|Y_v|)$ and $S = T[k - s^*\,..\,|X_v| + k - s^*)$. Since the Periodicity Test in Line 4 correctly returned $\textsc{Close}(Q)$, we have $\mathrm{HD}(Y_v, T) \leq \frac{1}{3r_v}$. In particular, this implies $\mathrm{HD}(Y_{v,s}, T[k - s\,..\,|X_v| + k - s)) \leq \frac{1}{3r_v}$ for every $s \in [-k\,..\,k]$. Similarly, since the Matching Test in Line 3 correctly returned $\textsc{Close}(s^*)$, we have $\mathrm{HD}(X_v, Y_{v,s^*}) \leq \frac{1}{3r_v}$. Consequently, $\mathrm{HD}(X_v, S) \leq \mathrm{HD}(X_v, Y_{v,s^*}) + \mathrm{HD}(Y_{v,s^*}, S) \leq \frac{2}{3r_v}$. By the triangle inequality, for every $s \in [-k\,..\,k]$, we have

$$\begin{aligned}\left|\mathrm{ED}(X_v, Y_{v,s}) - \mathrm{ED}(S, T[k-s\,..\,|X_v|+k-s))\right| \\ &\le \mathrm{ED}(X_v, S) + \mathrm{ED}(Y_{v,s}, T[k-s\,..\,|X_v|+k-s)) \\ &\le \mathrm{HD}(X_v, S) + \mathrm{HD}(Y_{v,s}, T[k-s\,..\,|X_v|+k-s)) \\ &\le \tfrac{2}{3r_v} + \tfrac{1}{3r_v} = \tfrac{1}{r_v}.\end{aligned}$$

Strings S and T both have a period $|Q| \le p$ and thus also a period $|Q| \cdot \lceil p/|Q| \rceil \in [p\,..\,2p)$. We apply Lemma 5.12 with the latter period, and return the approximation of $\mathrm{ED}(S, T[k-s\,..\,|X_v|+k-s))$ as $\eta_{v,s}$.

Since $|T| - |S| = 2k$, the running time is $O(k) + p\Delta \cdot (\log p)^{O(\log_\Delta p)}$, and each value $\eta_{v,s}$ is an $(\log p)^{O(\log_\Delta p)}$-approximation of $\mathrm{ED}(S, T[k-s\,..\,|X_v|+k-s))$, that is,

$$\mathrm{ED}(S, T[k-s\,..\,|X_v|+k-s)) \le \eta_{v,s} \le (\log p)^{O(\log_\Delta p)} \cdot \mathrm{ED}(S, T[k-s\,..\,|X_v|+k-s)).$$

Due to $\left|\mathrm{ED}(X_v, Y_{v,s}) - \mathrm{ED}(S, T[k-s\,..\,|X_v|+k-s))\right| \le \frac{1}{r_v}$, we conclude that

$$\mathrm{ED}(X_v, Y_{v,s}) - \tfrac{1}{r_v} \le \eta_{v,s} \le (\log p)^{O(\log_\Delta(p))} \cdot \left(\mathrm{ED}(X_v, Y_{v,s}) + \tfrac{1}{r_v}\right),$$

holds as claimed. ◻

Now we are ready to prove the main result of this section, which we restate for convenience.

Lemma 5.1 (Faster Algorithm for Bounded Block Periodicity)**.** *There exists an algorithm* AlgSmallBP *that, given two strings X, Y of length at most n, and parameters $k, K, p, B, \Delta \in \mathbb{Z}_+$ such that (i) $\mathrm{BP}_p(X) \le B$, (ii) $p, B \ge k$, (iii) $(16 \log K)^2 \le \Delta \le n$, and (iv) $K/k \ge (\log n)^{\beta \cdot \log_\Delta(n)}$ where $\beta > 0$ is a constant, solves the* GapED(k, K) *problem with probability at least 0.9 and runs in time*

$$O\left(\left(\frac{n}{K} \cdot \Delta + pB \cdot \Delta\right) \cdot (\log n)^{\alpha \cdot \log_\Delta(n)}\right)$$

for some constant $\alpha > 0$.

Setting Rates and Accuracies Recall that k, K, p and B are given parameters. Let T be a partition tree of degree ℓ for X, Y. For now, we keep $\ell \ge 2$ as a variable that will be set later. Note that T has at most $2n$ nodes in total, and its depth is bounded by $\lceil \log_\ell n \rceil$.

Let $\gamma = (\log p)^{c \log_\Delta(p)}$ where the constant $c > 0$ is chosen so that the approximation factor given by Lemma 5.13 does not exceed γ. We specify the rates and multiplicative accuracies for every node v in T in the following way:

- Multiplicative accuracy: if v is the root, we set $\alpha_v = 10 \cdot \gamma$. Otherwise, if v is a child of w, we set $\alpha_w = \alpha_v \cdot (1 - (2 \log n)^{-1})$. Note that since the depth of the tree is bounded by $\log n$, for every node v in T it holds that $\alpha_v \ge 10\gamma = \alpha_{\mathrm{root}}$.

- Rate: if v is the root, we set $r_v = 10000\gamma^2/K$. Otherwise, if v is a child of w then sample $u_v \sim \mathcal{D}(\varepsilon := (2\log n)^{-1}, \delta := 0.01 \cdot (kn)^{-1})$ (see Lemma 5.7) and set $r_v := r_w/u_v$.

Recall that our goal is to solve the tree distance problem, that is, for every node v compute values $\eta_{v,s}$ satisfying (5.5). We set the parameter μ in Definition 5.6 to $\mu := \gamma$.

Correctness We start by analyzing the correctness of Algorithm 10.

Lemma 5.14 (Correctness of Algorithm 10)**.** *Let X, Y be strings. Given any node v in the partition tree, Algorithm 10 correctly solves the Tree Distance Problem (Definition 5.6), with probability 0.9.*

Proof. We proceed by induction over the depth of the partition tree. That is, we will prove that for every node v, the algorithm correctly solves v (as per Definition 5.6) assuming that the recursive calls are correct.

For the base case, we consider all the nodes solved in Lines 1 to 7. If the node is solved in Lines 1 to 2, then we compute the tree distance exactly. If the node v is solved in Lines 5 to 7, then assuming that the calls to Lemma 5.13 succeed (we will bound the error probability later) the algorithm computes values $\eta_{v,s}$ satisfying

$$\mathrm{ED}(X_v, Y_{v,s}) - 1/r_v \le \eta_{v,s} \le \gamma \cdot \mathrm{ED}(X_v, Y_{v,s}) + \gamma/r_v.$$

We want to show that these values satisfy (5.5). Observe that the lower bound is satisfied (without additive error). For the upper bound, recall that $\alpha_v = 10\gamma \cdot (1 - (2\log n)^{-1})^d$, where $d \le \log n$ is the depth of v. In particular, observe that $\alpha_v \ge \gamma$. By Lemma 5.4, it holds that $\mathrm{ED}(X_v, Y_{v,s}) \le \mathrm{TD}^k(X_v, Y_{v,s})$. Therefore, we obtain $\eta_{v,s} \le \alpha_v \cdot \mathrm{TD}^k(X_v, Y_{v,s}) + \gamma/r_v$, as required (since $\mu = \gamma$).

For the inductive step, fix some node v and assume that all values $\eta_{v_i,s'}$ recursively computed in Line 10 are correct (i.e. they satisfy (5.5)). (We will bound the error probability later.) In Line 11, the algorithm computes a value $\widetilde{A}_{i,s}$ satisfying

$$\widetilde{A}_{i,s} = \min_{s' \in [-k\,..\,k]} \eta_{v_i,s'} + 2|s - s'| \le \min_{s' \in [-k\,..\,k]} \alpha_{v_i} \mathrm{TD}^k_{v_i,s'}(X, Y) + 2|s - s'| + \gamma/r_{v_i} \tag{5.10}$$

$$\widetilde{A}_{i,s} = \min_{s' \in [-k\,..\,k]} \eta_{v_i,s'} + 2|s - s'| \ge \min_{s' \in [-k\,..\,k]} 1/\alpha_{v_i} \mathrm{ED}^k_{v_i,s'}(X, Y) + 2|s - s'| - 1/r_{v_i} \tag{5.11}$$

Recall that for each child v_i the rate r_{v_i} is set to $r_{v_i} = r_v/u_{v_i}$, where u_{v_i} is an independent sample from the distribution $\mathcal{D}(\varepsilon = (2\log n)^{-1}, \delta = 0.01/(kn))$. Similarly, recall that the multiplicative accuracy α_{v_i} is set to $\alpha_{v_i} = \alpha_v(1 - (2\log n)^{-1})$ and is the same for all children.

We prove that the values $\eta_{v,s}$ computed in Line 13 satisfy the upper and lower bounds of (5.5):

Upper Bound Note that by Definition 5.3 it holds that

$$\mathrm{TD}^k_{v,s}(X,Y) = \sum_{i=0}^{\ell-1} A_{i,s}, \quad A_{i,s} = \min_{s' \in [-k\,..\,k]} \mathrm{TD}^k_{v_i,s'}(X,Y) + 2|s-s'|.$$

In particular, combining the above we obtain that $\widetilde{A}_{i,s} \le \alpha_v(1-(2\log n)^{-1})A_{i,s} + \gamma u_{v_i}/r_v$.

Therefore, we apply Lemma 5.7 (with $\alpha = \alpha_v(1-(2\log n)^{-1})$ and $\beta = \gamma/r_v$) to infer that in Line 13, the call to $\textsc{Recover}(\widetilde{A}_{0,s},\dots,\widetilde{A}_{\ell-1,s},u_{v_0},\dots,u_{v_{\ell-1}})$ computes a value satisfying (again, we assume that the output is correct and bound the error probability later)

$$\begin{aligned}\textsc{Recover}(\cdot) &\le (1+\varepsilon)\alpha\left(\sum_{i=0}^{\ell-1} A_{i,s}\right) + \gamma/r_v \\ &= (1+(2\log n)^{-1})\alpha_v(1-(2\log n)^{-1})\,\mathrm{TD}^k_{v,s}(X,Y) + \gamma/r_v \\ &\le \alpha_v\,\mathrm{TD}^k_{v,s}(X,Y) + \gamma/r_v,\end{aligned}$$

which concludes the proof for the upper bound.

Lower Bound Let $s_0,\dots,s_{\ell-1}$ be the shifts chosen by the algorithm in Line 11. Thus, from (5.11) we have that $\widetilde{A}_{i,s} \ge 1/\alpha_{v_i}(\mathrm{ED}(X_{v_i}, Y_{v_i,s_i}) + 2|s-s_i|) - 1/r_{v_i}$. Similarly as for the upper bound, suppose that the call to Recover in Line 13 succeeds. Then, by Lemma 5.7 (instantiated with parameters $\alpha = \alpha_v(1-(2\log n)^{-1})$ and $\beta = 1/r_v$) the call to $\textsc{Recover}(\widetilde{A}_{0,s},\dots,\widetilde{A}_{\ell-1,s},u_{v_0},\dots,u_{v_{\ell-1}})$ satisfies

$$\begin{aligned}\textsc{Recover}(\cdot) &\ge \frac{1}{(1+\varepsilon)\alpha}\left(\sum_{i=0}^{\ell-1}\mathrm{ED}(X_{v_i},Y_{v_i,s_i}) + 2|s-s_i|\right) - 1/r_v \\ &\ge \frac{1}{(1+\varepsilon)\alpha}\left(\sum_{i=0}^{\ell-1}\mathrm{ED}(X_{v_i},Y_{v_i,s_i}) + \mathrm{ED}(Y_{v_i,s},Y_{v_i,s_i})\right) - 1/r_v \\ &\ge \frac{1}{(1+\varepsilon)\alpha}\left(\sum_{i=0}^{\ell-1}\mathrm{ED}(X_{v_i},Y_{v_i,s})\right) - 1/r_v \\ &\ge \frac{1}{(1+\varepsilon)\alpha}\,\mathrm{ED}(X_v,Y_{v,s}) - 1/r_v\end{aligned}$$

The second inequality holds because we can transform $Y_{v_i,s}$ into Y_{v_i,s_i} by inserting and deleting $|s-s_i|$ characters. The third one by the triangle inequality. And the last inequality by the subadditivity of edit distance.

Finally, recall that $\alpha = \alpha_v(1-(2\log n)^{-1})$ and $\varepsilon = (2\log n)^{-1}$. Hence,

$$\frac{1}{(1+\varepsilon)\alpha} = \frac{1}{(1+(2\log n)^{-1})(1-(2\log n)^{-1})\alpha_v} \ge \frac{1}{\alpha_v},$$

which concludes the lower bound.

Error Probability There are four sources of randomness in the algorithm: the Matching and Periodicity tests in lines Lines 3 and 4, the call to Lemma 5.13 in Line 7, and the call to the recovery algorithm of precision sampling (Lemma 5.7) in Line 13. For each node, at least one of the Matching and Periodicity tests fails with probability at most $2\delta = 0.02/n$. For every leaf, the call to Lemma 5.13 fails with probability $0.01/n$. We apply the recovery algorithm of Lemma 5.7 with $\delta = 0.01/(kn)$ for $2k$ shifts in every node, hence the error probability is bounded by $0.02/n$ as well. By a union bound, the error probability for one node is bounded by $0.05/n$. Since there are at most $2n$ nodes in the tree, the total error probability is bounded by 0.1. □

Running Time Analysis We say that a node v in the partition tree is active if the recursive computation of Algorithm 10 reaches v.

Recall that we say that a node v in the partition tree is matched if there is a shift $s^* \in [-k\,..\,k]$ such that $X_v = Y_{v,s^*}$. If v is not matched, we call it unmatched. The following lemma shows that if $\mathrm{ED}(X, Y) \le k$, then there are not many unmatched nodes.

Lemma 5.15 (Number of Unmatched Nodes)**.** *Assume that* $\mathrm{ED}(X, Y) \le k$*. If the partition tree has depth* D*, then there are at most* kD *nodes which are not matched.*

Proof. Focus on any level in the partition tree and let $0 = i_0 < \cdots < i_w = n$ denote the partition induced by that level, i.e., let $[i_\ell\,..\,i_{\ell+1}] = I_v$ where v is the ℓ-th node in the level (from left to right). Let A be an optimal alignment between X and Y, then:

$$\mathrm{ED}(X, Y) = \sum_{\ell=0}^{w-1} \mathrm{ED}(X[i_\ell\,..\,i_{\ell+1}], Y[A(i_\ell)\,..\,A(i_{\ell+1})]).$$

Since we assumed that $\mathrm{ED}(X, Y) \le K$, there can be at most K nonzero terms in the sum. For any zero term we have that $X[i_\ell\,..\,i_{\ell+1}] = Y[A(i_\ell)\,..\,A(i_{\ell+1})]$ and therefore the ℓ-th node in the current level is matched with shift $A(i_\ell) - i_\ell$. By Proposition 4.10 we have that $|A(i_\ell) - i_\ell| \le \mathrm{ED}(X, Y) \le K$. This completes the proof. □

Lemma 5.16 (Number of Active Nodes)**.** *Suppose that* $\mathrm{ED}(X, Y) \le k$*. If the partition tree has depth* D*, then the number of active nodes is* $O(kD\ell + \mathrm{BP}_p(X)D\ell)$ *with probability at least 0.96.*

Proof. Recall that there are at most $2n$ nodes in the partition tree. Hence, by a union bound, all Matching Tests in Line 3 succeed with probability at least $1 - 0.02 = 0.98$. Similarly, all Periodicity Tests in Line 4 succeed with probability at least 0.98. Thus, by a union bound, all Matching Tests and Periodicity Tests succeed with probability at least 0.96. We condition on this event for the rest of the proof.

There are 3 kinds of active nodes v: (i) v and all its ancestors are unmatched, (ii) v is matched and all its ancestors are unmatched, and (iii) some ancestor of v (and thus v itself) is matched.

By Lemma 5.15, the number of nodes of type (i) is $O(kD)$. A node of type (ii) has a parent of type (i), so there are $O(kD\ell)$ such nodes in total. Finally, there can be at most $O(\mathrm{BP}_p(X)D\ell)$ nodes of type (iii) in total. To see this, observe that such a node will be active only if the periodicity test failed at its parent node. But by definition of block periodicity (Definition 4.7), at most $O(\mathrm{BP}_p(X))$ nodes per level fail the periodicity test. ◻

Lemma 5.17 (Running time of Algorithm 10)**.** *Let X, Y be strings with $k \le p$, $k \le \mathrm{BP}_p(X) \le B$ and $\mathrm{ED}(X,Y) \le k$. Let $2 \le \Delta \le n$ be a parameter. Then, Algorithm 10 runs in time*

$$\left(\frac{n}{K}\Delta + pB\cdot\Delta\right)(\log n)^{O(\log_\Delta(n))}$$

with probability at least 0.9.

Proof. We will set the degree ℓ of the partition tree in terms of Δ later. First we bound the expected running time of a single execution of Algorithm 10, ignoring the cost of recursive calls.

- Lines 1 and 2: If v is a leaf, then $|X_v| = 1$. Therefore, computing $\eta_{v,s}$ for each s takes $O(1)$ time. Thus, the overall time is $O(k)$.

- Lines 3 and 4: Running the Matching and Periodicity Tests (Lemmas 5.10 and 5.11) takes time $O(r_v|X_v|\log n + k\log|X_v| + p) \le O(r_v|X_v|\log n + k\log n + p)$.

- Lines 5 to 7: Running Lemma 5.13 takes time

$$O(p(\log p)^{O(\log_\Delta(p))}\Delta + k) \le p(\log p)^{O(\log_\Delta(p))}\Delta.$$

  Here we used the assumption that $k \le p$.

- Lines 8 to 11: The loop runs for ℓ iterations. In each iteration, we sample a precision in Line 9 in expected $O(1)$-time, perform a recursive computation which we ignore here, and apply Lemma 5.8 in time $O(k)$. The total time is $O(k\ell)$.

- Lines 12 to 13: The loop runs for $O(k)$ iterations and in each iteration we apply the recovery algorithm from Lemma 5.7 with parameters $\varepsilon = \Omega(1/\log n)$ and $\delta \ge 1/\mathrm{poly}(n)$. Each execution takes time $O(\ell\varepsilon^{-2}\log(\delta^{-1})) = O(\ell\,\mathrm{polylog}(n))$. Hence, the total time is $O(k\ell\,\mathrm{polylog}(n))$.

Thus, the overall time for one execution is

$$O(r_v|X_v|\log n + k\ell\,\mathrm{polylog}(n) + p(\log p)^{O(\log_\Delta(p))}\Delta). \tag{5.12}$$

We will simplify this term by plugging in the (expected) rate r_v for any node v.

Recall that $r_v = 10000\gamma^2 \cdot (K \cdot u_{v_1} \dots u_{v_d})^{-1}$ where $v_0, v_1, \dots, v_d = v$ is the root-to-node path leading to v and each u_i is sampled from $\mathcal{D}(\varepsilon = (2\log n)^{-1}, \delta = 0.01\cdot(kn)^{-1})$, independently. Using the efficiency property of Lemma 5.7 with $N = 200n$, there exist events $\mathcal{E}_w$ happening each with probability $1 - 1/N$ such that

$$\mathbb{E}(1/u_w \mid \mathcal{E}_w) \le O(\varepsilon^{-2} \operatorname{polylog}(N, \delta^{-1}, \varepsilon^{-1})) \le \operatorname{polylog}(n).$$

Taking a union bound over all nodes in T (there are at most $2n$ many), the event $\mathcal{E} := \bigwedge_w \mathcal{E}_w$ happens with probability at least 0.99; we will condition on $\mathcal{E}$ from now on. Under this condition, we have:

$$\begin{aligned}\mathbb{E}(r_v \mid \mathcal{E}) &= \frac{10000 \cdot \gamma^2}{K} \prod_{i=1}^{d} \mathbb{E}(1/u_{v_i} \mid \mathcal{E}_{v_i}) \\ &\le \frac{\gamma^2 \cdot (\log n)^{O(d)}}{K} \le \frac{\gamma^2 \cdot (\log n)^{O(\log_\ell(n))}}{K}.\end{aligned}$$

Let c be a large enough constant so that in the bound above we have $(\log n)^{O(\log_\ell(n))} \le (\log n)^{c \log_\ell(n)}$. We now set $\ell = \lceil (\log n)^{c \log_\Delta(n)} \rceil$, so that $(\log n)^{c \log_\ell(n)} \le \Delta$. Therefore, $\mathbb{E}(r_v \mid \mathcal{E}) \le \gamma^2 \Delta / K$.

Finally, we can bound the total expected running time (conditioned on $\mathcal{E}$) summing (5.12) over all active nodes v:

$$\sum_v O\left(|X_v| \cdot \frac{\gamma^2 \cdot \Delta}{K} + k\ell \operatorname{polylog}(n) + p(\log p)^{O(\log_\Delta(p))} \Delta \right).$$

To bound the first term, we use that $\sum_w |X_w| = n$ whenever w ranges over all nodes on a fixed level in the partition tree, and thus $\sum_v |X_v| \le n \cdot D$ where v ranges over all nodes. Thus, recalling that $\gamma = (\log p)^{O(\log_\Delta(p))}$ we can bound the first term in the sum by

$$\frac{n}{K} \cdot \Delta \cdot (\log p)^{O(\log_\Delta(p))}.$$

The other terms get multiplied by the number of active nodes. By Lemma 5.16, the number of active nodes is $O(kD\ell + \mathrm{BP}_p(X) D\ell) \le O(BD\ell)$ with probability at least 0.96 (here we used that $\mathrm{BP}_p(X) \le B$ and $k \le B$). Conditioned on this, the expected time for the second and third terms is bounded by

$$O\left(k\ell \operatorname{polylog}(n) \cdot BD\ell + p(\log p)^{O(\log_\Delta(p))} \Delta \cdot BD\ell\right).$$

Using that $k \le p$, and that $D \le \log n$, this is bounded by

$$O(p(\log p)^{O(\log_\Delta(p))} \Delta \cdot B\ell^2 \operatorname{polylog}(n)) \le p(\log n)^{O(\log_\Delta(n))} \Delta \cdot B.$$

In the last step we used that $\ell = (\log n)^{O(\log_\Delta(n))}$ and $p \le n$.

Combining the above, we conclude that the overall expected running time is bounded by

$$\begin{aligned}\frac{n}{K} \cdot \Delta \cdot (\log p)^{O(\log_\Delta(p))} + p(\log n)^{O(\log_\Delta(n))}\Delta \cdot B \\ \leq \left(\frac{n}{K} \cdot \Delta + pB \cdot \Delta\right) \cdot (\log n)^{O(\log_\Delta(n))}.\end{aligned}$$

We conditioned on two events: The event $\mathcal{E}$ and the event that the number of active nodes is bounded by $O(BD\ell)$ (Lemma 5.16). Both happen with probability at least 0.96, thus the total success probability is at least 0.9. ◻

Proof of Lemma 5.1. We run Algorithm 10 with 10 times the time budget

$$\left(\frac{n}{K}\Delta + pB \cdot \Delta\right) (\log n)^{O(\log_\Delta(n))}$$

given by Lemma 5.17. If the algorithm exceeds the time budget, we interrupt the computation and return Far. By the guarantee of Lemma 5.17 and Markov's inequality, returning Far in this case is correct with probability at least 0.9.

If the algorithm terminates, then it computes a value $\eta = \eta_{r,0}$ where r is the root node of the partition tree. By Lemma 5.14, this value satisfies

$$\frac{\mathrm{ED}(X,Y)}{10\gamma} - \frac{0.0001K}{\gamma^2} \leq \eta \leq 10\gamma\,\mathrm{TD}^k(X,Y) + \frac{0.0001K}{\gamma}.$$

Recall that here, $\gamma = (\log p)^{O(\log_\Delta(p))}$ is the approximation factor of Lemma 5.13. If $\mathrm{ED}(X,Y) \leq k$, then by Lemma 5.4 we obtain that

$$\eta \leq 10\gamma \cdot 3D\ell\,\mathrm{ED}(X,Y) + \frac{0.0001K}{\gamma} \leq (\log n)^{O(\log_\Delta(n))} \cdot k + \frac{0.0001K}{\gamma}$$

where we used that the depth D of the tree satisfies $D \leq \log n$, that $\ell = (\log n)^{O(\log_\Delta(n))}$ and that $p \leq n$. Conversely, if $\mathrm{ED}(X,Y) > K$ then

$$\eta \geq \frac{\mathrm{ED}(X,Y)}{10\gamma} - \frac{0.0001K}{\gamma^2} > \frac{0.099K}{\gamma}.$$

Thus, to distinguish these two cases we need that

$$\frac{0.099K}{\gamma} > (\log n)^{O(\log_\Delta(n))} \cdot k + \frac{0.0001K}{\gamma}.$$

Using that $p \leq n$, this holds if $K/k > (\log n)^{\Theta(\log_\Delta(n))}$ for a sufficiently large hidden constant. ◻

Comments and digest. The algorithm just presented to prove Lemma 5.1 closely follows the structure of the BCFN algorithm of Theorems 4.1 and 4.2 [Bri+22b]. However, there are some important differences.[2] Notoriously, our presentation is simplified based on the fact that the algorithm we gave can distinguish a gap which is a function of n, and not k (i.e. $K/k > (\log n)^{\Theta(\log_\Delta(n))}$ is a condition of Lemma 5.1). In the BCFN algorithm we had to do some more work to deal the case when $k \ll n$; note that the issue in this case, is that an algorithm solving e.g. the $(k, k \cdot n^{o(1)})$-gap problem is not sufficient to solve the desired $(k, k^{1+o(1)})$-gap. In particular, in [Bri+22b] we had to do some technical tricks, like stopping the depth of the recursion at depth $\mathrm{polylog}(k)$, which complicated the presentation and added some technical annoyances. The reason we can afford to do these simplifications here (i.e. obtain a gap as a function of n instead of k), is that to prove our Corollaries 4.4 and 4.5, we make use of Theorems 4.1 and 4.2 as blackboxes whenever $k \ll n$ (see Section 6.3).

5.3.1 Fast Edit Distance Approximation for Periodic Strings

In this section, we prove Lemma 5.12. Our main tool is the following recent result:

Theorem 5.18 (Dynamic Approximate ED [KMS23, Theorem 7.10])**.** *There exists a dynamic algorithm that, initialized with integers $2 \le b \le n$, maintains (initially empty) strings $X, Y \in \Sigma^{\le n}$ subject to character edits and, after each update, outputs an $O(b \log_b n)$-approximation of $\mathrm{ED}(X, Y)$. The amortized expected cost of each update is $b^2 \cdot (\log n)^{O(\log_b n)}$, and each answer is correct with high probability. The probabilistic guarantees hold against an oblivious adversary.*

First, let us change the parametrization to be consistent with the remainder of this chapter.

Corollary 5.19. *There exists a dynamic algorithm that, initialized with integers $2 \le \Delta \le n$, maintains (initially empty) strings $X, Y \in \Sigma^{\le n}$ subject to character edits and, after each update, outputs an $(\log n)^{O(\log_\Delta n)}$-approximation of $\mathrm{ED}(X, Y)$. The amortized expected cost of each update is $\Delta \cdot (\log n)^{O(\log_\Delta n)}$, and each answer is correct with high probability. The probabilistic guarantees hold against an oblivious adversary.*

Proof. Let $b = (\log n)^{c \log_\Delta n}$, where the constant c is chosen so that the update time of Theorem 5.18 does not exceed $b^2 \cdot (\log n)^{c \log_b n}$.

If $b \le n$, we simply use the algorithm of Theorem 5.18. Note that $\Delta = (\log n)^{c \log_b n}$, so the approximation ratio is $O(b \log_b n) = O((\log n)^{c \log_\Delta n} \cdot \log n) = (\log n)^{O(\log_\Delta n)}$, whereas the update time can be expressed as $b^2 \cdot (\log n)^{c \log_b n} = (\log n)^{2c \log_\Delta n} \cdot \Delta = \Delta \cdot (\log n)^{O(\log_\Delta n)}$.

If $b > n$, then $n \le (\log n)^{O(\log_\Delta n)}$, so it suffices to provide n-approximations of $\mathrm{ED}(X, Y)$ with $O(n)$ time per update. For this, we simply maintain X and Y, check

2. This paragraph is only relevant for those readers trying to figure out in which way our presentation here differs from [Bri+22b].

whether $X = Y$ for upon each update, and, depending on the outcome, report 0 or n as the n-approximation of $\mathrm{ED}(X, Y)$. ◻

We apply Corollary 5.19 to estimate the distances between a pattern P and substrings of a text T.

Lemma 5.20. *Given strings P and T of lengths $m \le n$, respectively, and integer $\Delta \in [2\,..\,n]$, one can compute, for all $i \in [0\,..\,n-m]$, multiplicative $(\log n)^{O(\log_\Delta n)}$-approximations of $\mathrm{ED}(P, T[i\,..\,i+m))$ in total time $n\Delta \cdot (\log n)^{O(\log_\Delta n)}$ correctly w.h.p.*

Proof. We apply Corollary 5.19 with the same parameters. Starting with empty X and Y, we use $2m$ edits to set $X := P$ and $Y := T[0\,..\,m)$. Next, we iteratively set $Y := T[i\,..\,i+m)$ for subsequent $i \in [0\,..\,n-m]$ to obtain $(\log n)^{O(\log_\Delta n)}$-approximations of $\mathrm{ED}(P, T[i\,..\,i+m))$. For this, we note that two edits (one insertion and one deletion) are enough to transform $T[i\,..\,i+m)$ to $T[i+1\,..\,i+m+1)$.

Overall, we use Corollary 5.19 for an oblivious sequence of $2n$ edits. Consequently, the total expected running time is $n\Delta \cdot (\log n)^{O(\log_\Delta n)}$ and all the answers are correct with high probability.

By Markov's inequality, the probability that the algorithm exceeds twice its expected time bound is $1/2$. By interrupting the algorithm after this time, and repeating the overall process $O(\log n)$ times, we obtain the lemma statement. ◻

Our next goal is to improve the running time provided that both strings have a common period. We start with an auxiliary combinatorial lemma that yields a 3-approximation of the edit distance between two p-periodic strings of the same length.

Lemma 5.21. *Let P, Q be strings of positive length p, let n be a positive integer, and denote $d = \lfloor n/p \rfloor$ and $r = n \bmod p$. Then, the edit distance of strings $X = P^*[0\,..\,n)$ and $Y = Q^*[0\,..\,n)$ satisfes*

$$\mathrm{ED}(X, Y) \le \mathrm{ED}(P[0\,..\,r), Q[0\,..\,r)) + \min_{s\in\mathbb{Z}}(d \cdot \mathrm{ED}(P, Q^{\circlearrowleft s}) + 2|s|) \le 3\,\mathrm{ED}(X, Y).$$

Proof. Let us start with the lower bound. Fix $s \in \mathbb{Z}$ and observe that the triangle inequality and the sub-additivity of edit distance yield

$$\begin{aligned}\mathrm{ED}(X, Y) &\le \mathrm{ED}(P^d, Q^d) + \mathrm{ED}(P[0\,..\,r), Q[0\,..\,r))\\ &\le \mathrm{ED}(P^d, (Q^{\circlearrowleft s})^d) + \mathrm{ED}(Q^d, (Q^{\circlearrowleft s})^d) + \mathrm{ED}(P[0\,..\,r), Q[0\,..\,r))\\ &\le d \cdot \mathrm{ED}(P, Q^{\circlearrowleft s}) + 2|s| + \mathrm{ED}(P[0\,..\,r), Q[0\,..\,r)).\end{aligned}$$

To prove upper bound, let us partition $Y = \bigodot_{i=0}^{d} Y[y_i\,..\,y_{i+1})$ so that

$$\mathrm{ED}(X, Y) = \sum_{i=0}^{d-1} \mathrm{ED}(P, Y[y_i\,..\,y_{i+1})) + \mathrm{ED}(P[0\,..\,r), Y[y_d\,..\,n)).$$

Fix $j \in [0\,..\,d)$ such that $\mathrm{ED}(P, Y[y_j\,..\,y_{j+1}))$ does not exceed the average

$$\tfrac{1}{d}\left(\mathrm{ED}(X,Y) - \mathrm{ED}(P[0\,..\,r), Y[y_d\,..\,n))\right).$$

Take $s := y_j - pj$ so that $Q^{\circlearrowleft s} = Y[y_j\,..\,y_j + p)$. Thus, by the triangle inequality,

$$\begin{aligned}\mathrm{ED}(P, Q^{\circlearrowleft s}) &\le \mathrm{ED}(P, Y[y_j\,..\,y_{j+1})) + \mathrm{ED}(Y[y_j\,..\,y_{j+1}), Y[y_j\,..\,y_j+p))\\ &= \mathrm{ED}(P, Y[y_j\,..\,y_{j+1})) + |y_{j+1} - y_j - p|\\ &\le 2\,\mathrm{ED}(P, Y[y_j\,..\,y_{j+1}))\\ &\le \tfrac{2}{d}\left(\mathrm{ED}(X,Y) - 2\,\mathrm{ED}(P[0\,..\,r), Y[y_d\,..\,n))\right).\end{aligned}$$

Similarly, $Q[0\,..\,r) = Y[pd\,..\,n)$ implies

$$\begin{aligned}\mathrm{ED}(P[0\,..\,r), Q[0\,..\,r)) &\le \mathrm{ED}(P[0\,..\,r), Y[y_d\,..\,n)) + \mathrm{ED}(Y[pd\,..\,n), Y[y_d\,..\,n))\\ &\le \mathrm{ED}(P[0\,..\,r), Y[y_d\,..\,n)) + |y_d - pd|\\ &\le 2\,\mathrm{ED}(P[0\,..\,r), Y[y_d\,..\,n)).\end{aligned}$$

At the same time,

$$2|s| = 2|y_j - pj| \le \mathrm{ED}(X[0\,..\,pj), Y[0\,..\,y_j)) + \mathrm{ED}(X[pj\,..\,n), Y[y_j\,..\,n)) = \mathrm{ED}(X,Y).$$

Overall, we have

$$\begin{aligned}&\mathrm{ED}(P[0\,..\,r), Q[0\,..\,r)) + d\cdot\mathrm{ED}(P, Q^{\circlearrowleft s}) + 2|s|\\ &\qquad\le 2\,\mathrm{ED}(P[0\,..\,r), Y[y_d\,..\,n))\\ &\qquad\qquad + d\cdot\frac{2}{d}(\mathrm{ED}(X,Y) - \mathrm{ED}(P[0\,..\,r), Y[y_d\,..\,n)))\\ &\qquad\qquad + \mathrm{ED}(X,Y)\\ &\qquad = 3\,\mathrm{ED}(X,Y),\end{aligned}$$

completing the proof. □

We are now ready to prove Lemma 5.12, whose statement is repeated below for readers' convenience.

Lemma 5.12. *Given a positive integer p, two strings P and T with period p and lengths $m \le n$, respectively, and an integer $\Delta \in [2\,..\,p]$, one can compute, for all $i \in [0\,..\,n-m]$, multiplicative $(\log p)^{O(\log_\Delta p)}$-approximations of $\mathrm{ED}(P, T[i\,..\,i+m))$ in time $O(n-m) + p\Delta\cdot(\log p)^{O(\log_\Delta p)}$ correctly w.h.p.*

Proof. Let us first argue that, without loss of generality, we can change the length of T to $m + p - 1$ while preserving its string period $T[0\,..\,p)$.

- If $n < m + p - 1$, then the distances $\mathrm{ED}(P, T[i\,..\,i+m))$ for $i \in [0\,..\,n-m]$ are unaffected by the extension. Moreover, extending T does not affect the claimed asymptotic runtime because the $O(n-m)$ term is dominated by $O(p)$.

- if $n > m + p - 1$, then the distances $\mathrm{ED}(P, T[i\,..\,i+m))$ for $i \in [0\,..\,p)$ are unaffected by the reduction. The remaining distances satisfy $\mathrm{ED}(P, T[i\,..\,i+m)) = \mathrm{ED}(P, T[i - p\,..\,i - p + m))$, so the approximation of the latter distance can be returned as the approximation of the former. This post-processing costs $O(n - m)$ extra time.

Thus, we henceforth assume $n = m + p - 1$.

We apply Lemma 5.20 twice: for $P[0\,..\,p)$ and $T[0\,..\,2p-1)$, as well as for $P[0\,..\,r)$ and $T[0\,..\,p+r-1)$, where $r = m \bmod p$. Suppose that, for each $i \in [0\,..\,p)$, this yields values e_i and f_i that are $(\log p)^{O(\log_\Delta p)}$-approximations of $\mathrm{ED}(P[0\,..\,p), T[i\,..\,i+p))$ and $\mathrm{ED}(P[0\,..\,r), T[i\,..\,i+r))$, respectively. For every $i \in [0\,..\,p)$, our algorithm returns

$$g_i := f_i + \min_{j \in [0\,..\,p)} (\lfloor m/p \rfloor \cdot e_j + 2\min(|i-j|, p - |i-j|)).$$

The values g_i can be computed as u-to-w_i distances in a weighted graph G consisting of $2p + 1$ vertices $u, v_0, \ldots, v_{p-1}, w_0, \ldots, w_{p-1}$ and the following edges for each $i \in [0\,..\,p)$:

- $u \longrightarrow v_i$ of length $\lfloor m/p \rfloor \cdot e_i$;
- $v_i \leftrightarrow v_{(i+1) \bmod p}$ of length 2;
- $v_i \longrightarrow w_i$ of length f_i.

This is because the shortest path of the form $u \longrightarrow v_j \rightsquigarrow v_i \longrightarrow w_i$ is of length precisely $\lfloor m/p \rfloor \cdot e_j + 2\min(|i-j|, p - |i-j|) + f_i$.

The running time is $p\Delta \cdot (\log p)^{O(\log_\Delta p)}$ for the applications of Lemma 5.20 plus $O(p \log p)$ for Dijkstra's single-source shortest paths algorithm. The first of these terms dominates.

The correctness stems from Lemma 5.21, which implies that, for every $i \in [0\,..\,p)$, the following quantity is a 3-approximation of $\mathrm{ED}(P, T[i\,..\,i+m))$:

$$\begin{aligned}&\mathrm{ED}(P[0\,..\,r), T[i\,..\,i+r))\\ &\qquad + \min_{j \in [0\,..\,p)} \{\lfloor m/p \rfloor \cdot \mathrm{ED}(P[0\,..\,p), T[j\,..\,j+p)) + 2\min(|i-j|, p - |i-j|)\}\end{aligned}$$

If we replace the two edit distances by their $(\log p)^{O(\log_\Delta p)}$-approximations, we get a $3 \cdot (\log p)^{O(\log_\Delta p)}$-approximation of $\mathrm{ED}(P, T[i\,..\,i+m))$. $\square$

6 Faster Sublinear Algorithm

The goal of this chaper is to prove our Main Theorem 4.3.

Organization The outline is as follows: In Section 6.1 we introduce the crucial notion of *breaks* and their usage as a proxy for the block periodicity. In Section 6.2 we give our main algorithm, in which we leverage the result from the previous chapter (Lemma 5.1). Finally, we put things together in Section 6.3 where we prove our main result and its corollaries.

6.1 Breaks and their Usage

We start by formally defining breaks.

Definition 6.1 (Break)**.** *Let X be a string of length n and let $k \geq 2$ be an integer. A position $i \in [0\,..\,n-3k]$ is a k-*break *in X if i is a multiple of k and* $\mathrm{per}(X[i\,..\,i+3k)) > k$*, that is, $X[i\,..\,i+3k)$ is* not *k-periodic.*

The importance of breaks is that their number is a good approximation of the block periodicity of a string, as captured by the following lemma.

Lemma 6.2. *Let b be the number of k-breaks in X. Then, $\frac{1}{3}b \leq \mathrm{BP}_k(X) \leq b+3$.*

Proof. We first argue that $\mathrm{BP}_k(X) \leq b+3$. For this, we partition X into (at most) $b+3$ pieces, splitting X at position $i+2k$ for every break i, as well as at positions k and $n-k$. It suffices to prove that each of the resulting pieces has period at most k. This property is trivially satisfied for pieces contained within $X[0\,..\,k)$ and $X[n-k\,..\,n)$ (their length does not exceed k). So, consider a piece $X[p\,..\,q)$ with $k \leq p < q \leq n-k$ and let $r \in [p\,..\,q]$ be the largest position such that $\mathrm{per}(X[p\,..\,r)) \leq k$. For a proof by contradiction, suppose that $r < q$. Consider an integer $i \in (r-3k\,..\,r-2k]$ that is a multiple of k. Observe that $r \geq p+k \geq 2k$ implies $i \geq 0$ and $r < q \leq n-k$ implies $i \leq n-3k$. Moreover, there is no piece starting at position $i+2k \in (r-k\,..\,r] \subseteq (p\,..\,r]$, and thus i is not a break. This means that $\mathrm{per}(X[i\,..\,i+3k)) \leq k$. If $i \leq p$, then $\mathrm{per}(X[p\,..\,i+3k]) \leq \mathrm{per}(X[i\,..\,i+3k)) \leq k$, contradicting the choice of $r < i+3k$. Otherwise, the intersection $X[i\,..\,r)$ of $X[i\,..\,i+3k)$ and $X[p\,..\,r)$ contains at least $2k$ characters. By the periodicity lemma [FW65], since both fragments have periods at most k, their union $X[p\,..\,i+3k)$ also has period at most k, contradicting the choice of r.

Next, we argue that $b \leq 3\,\mathrm{BP}_k(X)$. Let $L := \mathrm{BP}_k(X)$. By definition, we can write X as $X = X_1 X_2 \cdots X_L$, where each X_i is k-periodic. Observe that every substring inside some X_i is k-periodic, so no break can be contained inside any X_i. In particular, every break contains the starting position of at least some X_i. Moreover, the starting position of any X_i is contained in at most three breaks (since breaks overlap). Therefore, the number of breaks is bounded by $3L$. ◻

At the heart of our algorithm is the following splitting algorithm. The idea is to use breaks in order to split an instance into independent subproblems, where each subproblem has smaller block periodicity.

Lemma 6.3 (Splitting)**.** *There is an algorithm* Split(X, Y, k, K, δ) *that, given as input two length-n strings X, Y, two thresholds $1 \leq k \leq K$, and a parameter $\delta \in (0, 1]$, returns partitions $X = X_1 \cdots X_s$ and $Y = Y_1 \cdots Y_s$ such that:*

1. *With probability at least $1-\delta$, the inequality $\mathrm{BP}_k(X_i) \leq 4K$ is satisfied for all $i \in [1\,..\,s]$.*
2. *If $\mathrm{ED}(X, Y) \leq k$, then $\sum_{i=1}^{s} \mathrm{ED}(X_i, Y_i) = \mathrm{ED}(X, Y)$ holds with probability at least $1 - \frac{3k}{K}\log\frac{n}{\delta}$.*

The algorithm runs in expected time $O\left(\frac{n}{K}\log\frac{n}{\delta}\right)$.

Proof. We sample indices $i \in [0\,..\,n-3k]$ which are multiples of k uniformly at random with rate $\frac{1}{K}\log\frac{n}{\delta}$. We identify all k-breaks in X among the sampled indices. For each k-break i, we try to find a position $j \in [i - \lfloor k/2 \rfloor\,..\,i + \lfloor k/2 \rfloor]$ such that $X[i\,..\,i+3k) = Y[j\,..\,j+3k)$. Since i is a break, there is at most one such position j. If there are none, we set $j := i$. In either case, we split X and Y at positions i and j, respectively. See Algorithm 11 for the pseudocode.

Let us first analyze the running time. For each index i, computing $\mathrm{per}(X[i\,..\,i+3k))$ and finding the occurrences of $X[i\,..\,i+3k)$ within $Y[i-\lfloor k/2\rfloor\,..\,i+3k+\lfloor k/2\rfloor)$ takes $O(k)$ time since period finding and pattern matching are in linear time [KJP77]. In expectation, the number of sampled indices $i \in [0\,..\,n-3k]$ that are multiples of k is $O(\frac{n}{k}\cdot\frac{1}{K}\log\frac{n}{\delta})$. Therefore, the algorithm runs in $O(\frac{n}{K}\log\frac{n}{\delta})$ expected time, as desired.

Next, let us analyze the correctness. For property (1), note that it suffices to show that we sample at least one k-break out of every K consecutive k-breaks. Indeed, if this claim holds, then, by Lemma 6.2, we obtain that every phrase X_i satisfies $\mathrm{BP}_k(X_i) \leq K+3 \leq 4K$. We proceed to bound the probability of this event. Fix K consecutive breaks. The probability that we do not sample any of them is

$$\left(1 - \frac{\log(n/\delta)}{K}\right)^K \leq \exp(-\log(n/\delta)) = \frac{\delta}{n}.$$

By a union bound over the at most n choices of K consecutive k-breaks, we get property (1).

For property (2), suppose that $\mathrm{ED}(X, Y) \leq k$ and fix an optimal alignment. Then, the optimal alignment performs at least one edit in at most $3k$ fragments $X[i\,..\,i+3k)$

Algorithm 11

```
procedure Split($X, Y, k, K, \delta$)
    Sample a set $S \subseteq [0\,..\,n-3k]$ of multiples of $k$
        including each element independently with probability $\frac{1}{K}\log\frac{n}{\delta}$
    Let $i_1 < i_2 < \cdots < i_{s-1}$ be the $k$-breaks in $S$
    for $\ell \in [1\,..\,s)$ do
        if there exists $z \in [i_\ell - \lfloor k/2 \rfloor\,..\,i_\ell + \lfloor k/2 \rfloor]$
            such that $X[i_\ell\,..\,i_\ell + 3k) = Y[z\,..\,z+3k)$ then
            $j_\ell := z$
        else
            $j_\ell := i_\ell$
    Set $i_0 := j_0 := 0$ and $i_s := |X|, j_s := |Y|$
    Set $X_\ell := X[i_{\ell-1}\,..\,i_\ell), Y_\ell := [j_{\ell-1}\,..\,j_\ell)$ for $\ell \in [1\,..\,s]$
    return $X_1, Y_1, \dots, X_s, Y_s$
```

where $i \in [0\,..\,n-3k)$ is a multiple of k. Thus, the probability that we do not sample any such index i is at least

$$\left(1 - \frac{\log(n/\delta)}{K}\right)^{3k} \geq 1 - \frac{3k}{K}\log\frac{n}{\delta}.$$

Condition on this event. This means that the fixed optimal alignment matches the sampled breaks perfectly to fragments of Y. Moreover, since they are breaks, they have a unique perfect match in Y. This implies the claim. □

6.2 Algorithm

We are now ready to present our algorithm. Since this is the most technically involved part of our work, we start with an informal overview to convey some intuition.

Algorithm Overview The high-level idea of the algorithm is as follows: If the block periodicity of X is bounded, say $\mathrm{BP}_k(X) \leq K$, then we can solve the GapED(k, K) problem directly using Lemma 5.1. Otherwise, we apply Lemma 6.3 to split the strings into pieces $X_1, Y_1, \dots, X_s, Y_s$ so that $\mathrm{ED}(X,Y) = \sum_i \mathrm{ED}(X_i, Y_i)$. It remains to distinguish whether $\mathrm{ED}(X,Y) = \sum_i \mathrm{ED}(X_i, Y_i) < k$, or $\mathrm{ED}(X,Y) = \sum_i \mathrm{ED}(X_i, Y_i) > K$. Split guarantees that, with good probability, the block periodicity of each piece is bounded. Hence, we naturally want to recurse on some of these pieces. However, we cannot afford to naively recurse on all them since their total size is too large, and we are aiming for sublinear time. This is exactly the task where we can apply the precision sampling technique, which enables us to recurse in a few subproblems of total length $\approx n/K$. To obtain our desired running time, we additionally need that the recursive calls run in time proportional to their edit distance $\mathrm{ED}(X_i, Y_i)$, so that we can *distribute* our time budget among them.

To implement the ideas above, we need to overcome some obstacles. Importantly, observe that Split (see Lemma 6.3) guarantees that $\sum_i \mathrm{ED}(X_i, Y_i) = \mathrm{ED}(X, Y)$ holds with good probability *only in the case when* $\mathrm{ED}(X, Y) \leq k$. If $\mathrm{ED}(X, Y) > k$ or Split fails, then all we know is that the split satisfies $\sum_i \mathrm{ED}(X_i, Y_i) \geq \mathrm{ED}(X, Y)$ due to subadditivity of edit distance. Moreover, note that a priori there is no way to know whether $\sum_i \mathrm{ED}(X_i, Y_i) = \mathrm{ED}(X, Y)$ holds. With this in mind, we design two routines AlgMain and AlgBoosted, see Algorithm 12 for the pseudocode. These routines receive as inputs strings X, Y, thresholds $k < K$ and a parameter p with the promise that $\mathrm{BP}_p(X) \leq K/k \cdot p$ (additionally, AlgBoosted receives a parameter δ that determines its failure probability). Intuitively, AlgMain solves the $\mathrm{GapED}(k, K)$ problem and runs in the desired running time[1] $\widehat{O}(n/K + kK)$ *if we have the promise* that $\mathrm{ED}(X, Y) \leq k$ (i.e. in the Close case). AlgBoosted solves the $\mathrm{GapED}(k, K)$ problem with high probability and runs in time $\widehat{O}(n/K + \min(k, \mathrm{ED}(X, Y)) \cdot K)$. Crucially, one of the terms in the running time of AlgBoosted is proportional to $\mathrm{ED}(X, Y)$ (this allows us to distribute the time budget among subproblems) and unlike AlgMain, it does not assume that $\mathrm{ED}(X, Y) \leq k$.

AlgMain carries out the ideas described above—namely, if the block periodicity is bounded then we solve the problem directly using Lemma 5.1 in Line 13 (after handling some trivial base cases in Lines 8 to 11). Otherwise, we call Split in Line 15 and perform precision sampling in Lines 16 to 21. In order to distribute the time budget among the sampled subproblems, we make calls to AlgBoosted in Line 18. AlgBoosted in turn solves the $\mathrm{GapED}(k, K)$ by calling to AlgMain. In order to run in time $\widehat{O}(n/K + \min(k, \mathrm{ED}(X, Y))K)$, we use exponential search in Lines 2 to 4. Since AlgMain only satisfies its running time guarantee when we are in the Close case, each call in Line 4 needs to be stopped and interrupted if it exceeds this time budget. Note that each time that AlgMain calls AlgBoosted, the block periodicity is reduced so the recursion eventually stops.

Running Time and Correctness Analysis Formally, we prove the following lemma:

Lemma 6.4. *There is an algorithm that given strings X, Y of total length n, and parameters k, K, p and $\Delta \in \mathbb{Z}_+$ such that: (i) $p \leq n$, (ii) $\mathrm{BP}_p(X) \leq \frac{K}{k} p$, (iii) $(256 \log K)^2 \leq \Delta \leq n$, and (iv) $K/k > (\log n)^{c \log_\Delta(n)}$ for a sufficiently large constant $c > 0$, solves the $\mathrm{GapED}(k, K)$ problem with probability at least $2/3$. The algorithm runs in time*

$$\left(\frac{n}{K}\Delta + \min(E, k) \cdot K\Delta^3\right) \cdot (\log n)^{O(\log_\Delta(n))},$$

where $E := \mathrm{ED}(X, Y)$.

1. For this informal overview, we use the notation $\widehat{O}(\cdot)$ to ignore subpolynomial factors $n^{o(1)}$ for the purpose of readability, the actual proof does not hide these factors.

Algorithm 12

```
1  procedure AlgBoosted($X, Y, k, K, p, \delta$)
2      for $\tilde{k} = 0, 1, 2, 4, \dots, k$ do
3          repeat $\Theta(\log(\log(k)/\delta))$ times
4              Run AlgMain($X, Y, \tilde{k}, K, p$) and store the outcome. If it
                   does not finish within time budget
                   $O\left(\left(\frac{n}{K}\Delta + \tilde{k}K \cdot \Delta^3\right) \cdot (\log n)^{\alpha \cdot \log_\Delta(n)} \cdot (\log n)^{14 \log_\Delta(p)}\right)$
                   interrupt the execution and store Far
5          if the majority of the outcomes is Close then return Close
6      return Far
7  procedure AlgMain($X, Y, k, K, p$)
8      if $K > |X| + |Y|$ then
9          return Close
10     if $k = 0$ then
11         return output of Equality Test (Lemma 5.9) for $X, Y$ with $r := 1/K, \delta := 0.001$
12     if $p \le k\Delta$ then
13         return AlgSmallBP($X, Y, k, K, k\Delta, K\Delta$) (Lemma 5.1)
14     else
15         Let $X_1, Y_1, \dots, X_s, Y_s$ be the output of Split($X, Y, k, K, 0.01$)
16         for $d = 1, 2, 4, \dots, K$ do
17             Sample a set $S \subseteq [1 \,..\, s]$ including each element
                   independently with probability $\frac{108 d \log K \log(1/\delta)}{K}$, where $\delta$ :=
   $0.01/\log K$
18             Run AlgBoosted($X_i, Y_i, d\frac{k}{K}64 \log K, d, 16k \log K, 0.01/n^2$) for all $i \in S$
19             if at least $12 \log(1/\delta)$ of the answers are Far then
20                 return Far
21         return Close
```

Proof. We prove the lemma using mutual induction over both AlgBoosted and AlgMain. Formally, we have the following inductive hypothesis.

Inductive Hypothesis Let X, Y be strings of total length n, and let $k, K, p, \Delta \in \mathbb{Z}_+$ be parameters such that: (i) $p \le n$, (ii) $\mathrm{BP}_p(X) \le \frac{K}{k} p$, (iii) $(256 \log K)^2 \le \Delta$, and (iv) and $K/k \ge (\log n)^{\beta \log_\Delta(n) + 14 \log_\Delta(p)}$, where $\beta > 0$ is the same constant as in Lemma 5.1. Then, the following holds:

(i) Let $E := \mathrm{ED}(X, Y)$. AlgBoosted(X, Y, k, K, p, δ) solves the GapED(k, K) problem with probability at least $1 - \delta$ and runs in time

$$O\left(\left(\frac{n}{K}\Delta + \min(E, k)K \cdot \Delta^3\right) (\log n)^{\alpha \cdot \log_\Delta(n)} \cdot (\log n)^{14 \log_\Delta(p)} \cdot \log(\log(k)/\delta) \log k\right).$$

(ii) With probability at least 0.9, AlgMain(X, Y, k, K, p) solves the GapED(k, K) problem. Moreover, if ED$(X, Y) \leq k$, then, with probability at least 0.9, it runs in time

$$O\left(\left(\frac{n}{K}\Delta + kK \cdot \Delta^3\right) \cdot (\log n)^{\alpha \cdot \log_\Delta(n)} \cdot (\log n)^{14 \log_\Delta(p)}\right).$$

For both running times above, $\alpha > 0$ is the constant in the running time of Lemma 5.1.

Base Case Let X, Y be strings of total length n and let k, K, p be integers satisfying the conditions of the inductive hypothesis. For the base case, we prove statement (ii) when $n < 4$, $K > n$, $k = 0$ or $p \leq k\Delta$. If $n < 4$, then we can solve the problem directly trivially in time $O(1)$ (we omit this trivial case from the pseudocode for brevity). If $K > n$, then we can return Close since ED$(X, Y) \leq n$. If $k = 0$, then the task is to distinguish whether $X = Y$ or ED$(X, Y) > K$. Since HD$(X, Y) \geq$ ED(X, Y), the Equality Test (see Lemma 5.9) correctly solves the problem in Line 11 in time $O(n/K)$; this is correct with probability 0.99. Otherwise, since $p \leq k\Delta$, the subproblem is solved directly in Line 13. By assumption, we know that BP$_p(X) \leq K/k \cdot p \leq K\Delta$ and that $K/k \geq (\log n)^{\beta \log_\Delta(n) + 14 \log_\Delta(p)}$. This means that the call to AlgSmallBP is valid. Hence, by Lemma 5.1, the base case is solved with probability 0.9 in overall time

$$O\left((n/K\Delta + kK\Delta^3)(\log n)^{\alpha \cdot \log_\Delta(n)}\right),$$

which satisfies (ii).

Inductive Step for AlgBoosted Let X, Y be strings of total length n and let k, K, p be integers satisfying the conditions of the inductive hypothesis. Suppose that (ii) holds for any parameters $n' \leq n, k' \leq k, K' \leq K, p' \leq p$ (which satisfy the conditions of the inductive hypothesis). We will prove that (i) holds for X, Y, k, K, p.

Consider the execution of AlgBoosted(X, Y, k, K, p, δ). The calls that it makes in Line 4 to AlgMain satisfy (ii) as we just argued. We start by showing correctness. First, suppose that $E :=$ ED$(X, Y) \leq k$. Consider any iteration of the loop in Line 2 with parameter $\tilde{k} \leq E$. Since $E \leq k$, returning Close is correct, so if the majority of outcomes is Close then we correctly return Close. If none of these iterations return Close, consider the iteration when $\tilde{k}/2 < E \leq \tilde{k}$. At this iteration, by (ii), each call to AlgMain correctly returns Close and runs in the time budget with probability at least 0.8. Since we take the majority outcome out of $\Theta(\log(\log(k)/\delta))$ repetitions, by Chernoff's bound, we conclude that for this iteration we return Close in Line 5 with probability at least $1 - \delta/\log k \geq 1 - \delta$.

Now, consider the case when $E > K$. Fix some iteration of the loop in Line 2. By (ii), for each call to AlgMain in Line 4, we correctly store Far with probability at least 0.9 (regardless of whether we interrupt the algorithm or not). Thus, by Chernoff's bound, we do not return Close in Line 5 with probability at least $1 - \delta/\log k$. By a union bound, we conclude that we do not return Close in any of $O(\log k)$ iterations of the loop in Line 2 with probability at least $1 - \delta$.

Now, we argue about the running time. As shown above, if $E \leq k$, then with probability at least $1 - \delta$ we return CLOSE in the iteration when $\tilde{k}/2 < E \leq \tilde{k}$. If $E > K$, then we execute all the $\log k$ iterations of the loop in Line 2. Thus, the overall running time is bounded by

$$O\left(\left(\frac{n}{K}\Delta + \min(E,k)K \cdot \Delta^3\right) \cdot (\log n)^{\alpha \cdot \log_\Delta(n)} \cdot (\log n)^{14 \log_\Delta(p)} \log(\log(k)/\delta) \cdot \log k\right).$$

Inductive Step for AlgMain Consider strings X, Y of total length n and let k, K, p be integers satisfying the conditions of the inductive hypothesis. Suppose that (i) holds for any parameters $n' \leq n, k' \leq k, K' \leq K$ and $p' < p$ (note that here p' is strictly less than p). We proceed to prove statement (ii) for X, Y, k, K, p.

The cases of $n < 4$, $K > n$, $k = 0$ or $p \leq k\Delta$ were handled by the base case, so assume that $n \geq 4$, $K \leq n$, $k \geq 1$ and $p > k\Delta$. Therefore, the algorithm continues in Lines 15 to 21 and makes recursive calls to ALGBOOSTED.

First, we analyze the running time. For this, we can assume that $E = \mathrm{ED}(X, Y) \leq k$. The call to SPLIT in Line 15 runs in expected time $O(\frac{n}{K} \log n)$. By Lemma 6.3, with probability at least $1 - \frac{3k}{K}\log(n/0.01)$, its output $X_1, Y_1, \ldots, X_s, Y_s$ satisfies $\sum_i \mathrm{ED}(X_i, Y_i) = E$, and $\mathrm{BP}_k(X_i) \leq 4K$ for every $i \in [1\,..\,s]$ with probability 0.99. By assumption, we have that $K/k \geq (\log n)^{\beta \log_\Delta(n) + 14 \log_\Delta(p)}$ where $\beta > 0$ is a sufficiently large constant. Thus, by a union bound, can bound the overall success probability by $1 - 3(k/K)\log(n/0.01) - 0.01 \geq 0.98$. From now on, we condition on this event. Let $E_i := \mathrm{ED}(X_i, Y_i)$ and $n_i := |X_i| + |Y_i|$.

▷ Claim 6.5. Every call to $\mathrm{ALGBOOSTED}(X_i, Y_i, k', K', p', \delta')$ in Line 18 satisfies:

(1) $p' \leq p/\Delta^{1/2}$ and $p' \leq n'$ where $n' = |X_i| + |Y_i|$,

(2) $\mathrm{BP}_{p'}(X_i) \leq (K'/k') \cdot p'$,

(3) $K'/k' \geq (\log n_i)^{\beta \log_\Delta(n_i) + 14 \log_\Delta(p')}$,

(4) $\Delta > (256 \log K')^2$.

Proof. Observe that, in Line 18, the algorithm sets $p' = 256k \log K$ and

$$\frac{K'}{k'} = \frac{d}{d(k/K)64 \log K} = \frac{K}{64k \log K}.$$

To show (2), note that since for each X_i it holds that $\mathrm{BP}_k(X_i) \leq 4K$, it follows that $\mathrm{BP}_{p'}(X_i) \leq 4K = (K'/k') \cdot p'$. Since $p > k\Delta$ and $\Delta > (256 \log K)^2$, we obtain that $p' < p/\Delta^{1/2}$. For the corner case when $256k \log K > n'$, note that setting $p' = \min(n', 256k \log K)$ we still have that $p' < p/\Delta^{1/2}$, and (2) remains valid too (we omitted this case from the pseudocode for readability). Therefore, we obtain (1).

To obtain (3), we use the assumption that $K/k \geq (\log n)^{\beta \log_\Delta(n) + 14 \log_\Delta(p)}$ and that $K \leq n$ (since the case $K > n$ was handled by the base case), yielding

$$K'/k' = K/(64k \log K) \geq (\log n)^{\beta \log_\Delta(n) + 14 \log_\Delta(p) - 1}/64.$$

We continue bounding this expression. By (1), we have that $\log_\Delta(p) \geq \log_\Delta(p'\Delta^{1/2}) = \log_\Delta(p') + 1/2$. Hence, using that $64 \leq (\log n)^6$ (the case $n < 4$ was handled by the base case) and $n_i = |X_i| + |Y_i| \leq n$, we obtain that $K'/k' \geq (\log n_i)^{\beta \log_\Delta(n_i) + 14\log_\Delta(p')}$.

Finally, (4) follows since $K' \leq K$ and by assumption $\Delta > (256 \log K)^2$. $\triangleleft$

To bound the expected running time of Lines 16 to 21, observe that, for a fixed value of d, the probability that a subproblem X_i, Y_i is called in Line 18 is $\frac{108d}{K} \log K \log(1/\delta)$, where $\delta = 0.01/\log K$. Let $E_i := \mathrm{ED}(X_i, Y_i)$ and $n_i := |X_i| + |Y_i|$. Claim 6.5 implies that we can use the inductive hypothesis (i) to bound the running time for each call to AlgBoosted. Hence, the expected running time of one iteration of the loop in Line 16 can be bounded as

$$O\left(\sum_{i=1}^{s} \frac{d}{K}\left(\frac{n_i}{d}\Delta + E_i d\Delta^3\right)(\log n_i)^{\alpha \log_\Delta(n_i)}(\log n_i)^{14\log_\Delta(p')} \log^2 n \log K \log(1/\delta)\right).$$

Here, we bounded the factors $\log(\log(k')/\delta')\log(k') \leq O(\log^2 n)$ in the running time of AlgBoosted given by (i), since the parameter δ' in the call to AlgBoosted is set to $\delta' = 0.01/n^2$, and $k' \leq K \leq n$.

By Claim 6.5 we have $p' \leq p/\Delta^{1/2}$, and therefore $14\log_\Delta(p') \leq 14\log_\Delta(p) - 5$. Since $n_i \leq n$, $K \leq n$, and $1/\delta \leq n^2$, we can bound

$$(\log n_i)^{14\log_\Delta(p')} \log^2 n \log K \log(1/\delta) \leq O((\log n)^{14\log_\Delta(p)-1}).$$

By the guarantees of split, we know that $\sum_i n_i = n$ and $\sum_i E_i = E$. Thus, combining the above, we can bound the expected running time of one iteration of the loop in Line 16 as

$$O\left(\left(\frac{n}{K}\Delta + EK\Delta^3\right)(\log n)^{\alpha \log_\Delta(n)}(\log n)^{14\log_\Delta(p)-1}\right),$$

where we bounded $Ed^2/K \leq EK$ using that $d \leq K$. The overall expected running time across the $O(\log K)$ iterations of the for loop in Line 16 adds another $O(\log K) \leq O(\log n)$ factor. Hence, we obtain expected time

$$O\left(\left(\frac{n}{K}\Delta + EK\Delta^3\right)(\log n)^{\alpha \log_\Delta(n)}(\log n)^{14\log_\Delta(p)}\right).$$

Finally, by Markov's inequality, the algorithm does not exceed 20 times this time bound with probability at least 0.95. Together with the initial conditioning on the success of Split, we obtain total success probability at least 0.9. This completes the proof of the running time for (ii).

Now we argue about the correctness of Lines 16 to 21. Due to Claim 6.5, we can use the inductive hypothesis on the calls to AlgBoosted in Line 18. Thus, each call is correct with probability at least $1 - 0.01/n^2 \geq 1 - 0.01/(s \log K)$ and hence, all the calls are correct with probability at least 0.99. The correctness of the algorithm follows from the following claim:

▷ Claim 6.6. Consider an execution of Lines 16 to 20. Suppose all recursive calls are correct. Let $E_i := \mathrm{ED}(X_i, Y_i)$ for $i \in [1\,..\,s]$. Let $\mathcal{E}$ be the event that, if $\sum_i E_i > K$, then we return FAR and, if $\sum_i E_i \leq k$, then we return CLOSE. Then, $\mathbb{P}(\mathcal{E}) \geq 0.99$.

Before proving the claim, let us see how to derive correctness from it. As argued in the running time analysis, if $\mathrm{ED}(X,Y) \leq k$ then with probability at least 0.99 the call to SPLIT succeeds, and we obtain $\sum_i E_i = \mathrm{ED}(X,Y) \leq k$ (see Lemma 6.3). Then, by Claim 6.6 we return CLOSE. To bound the error probability, we take a union bound over all the $O(s \log K)$ calls to ALGBOOSTED, the call to SPLIT, and Claim 6.6. All calls to ALGBOOSTED succeed with probability at least 0.99, the call to SPLIT succeeds with probability 0.99, and Claim 6.6 succeeds with probability 0.99. Thus, the overall success probability is at least 0.95. On the other hand, if $\mathrm{ED}(X,Y) > K$, then by the subadditivity of edit distance, the partition returned by SPLIT satisfies $\sum_i E_i \geq \mathrm{ED}(X,Y) > K$ (deterministically). Thus, assuming that the calls to ALGBOOSTED are correct, Claim 6.6 guarantees that we return FAR. Similarly as before, the success probability is at least 0.95. It remains to prove the claim.

Proof of Claim 6.6. First, consider the case where $\sum_i E_i > K$. For $\ell \geq 0$, define

$$L_\ell := \{\, j \mid 2^\ell \leq \min(E_j, K+1) < 2^{\ell+1} \,\}.$$

Thus, note that there are at most $T := \lfloor \log(K+1) \rfloor + 1$ non-empty sets L_ℓ. The key observation is that since $\sum E_i > K$, there exists a level ℓ^* such that $|L_{\ell^*}| \geq K/(2^{\ell^*+1}T)$. Indeed, note that otherwise we obtain

$$K \leq \sum_{j=1}^{s} \min(E_j, K+1) \leq \sum_{\ell=0}^{\lfloor \log(K+1) \rfloor} 2^{\ell+1} |L_\ell| < \sum_{\ell=0}^{\lfloor \log(K+1) \rfloor} 2^{\ell+1} \frac{K}{2^{\ell+1}T} = K;$$

a contradiction.

Fix such a level ℓ^* and focus on the iteration of the loop in Line 16 when $d = 2^{\ell^*}$. For each $i \in L_{\ell^*}$, let I_i be the indicator random variable which equals 1 if i is included in the sample S. Let $I := \sum_{i \in L_{\ell^*}} I_i$. Since we sample S with rate $\frac{108d}{K} \log K \log(1/\delta)$, it follows that

$$\mathbb{E}(I) = |L_{\ell^*}| \cdot \frac{108d}{K} \log K \log(1/\delta) \geq \frac{K}{2dT} \cdot \frac{108d}{K} \log K \log(1/\delta) = \frac{54}{T} \log K \log(1/\delta).$$

Since $T = \lfloor \log(K+1) \rfloor + 1 \leq \log(K) + 2 \leq 3 \log K$, we obtain that $\mathbb{E}(I) \geq 18 \log(1/\delta)$. Note that the algorithm correctly returns FAR if $I \geq 12 \log(1/\delta)$ (see Line 19). By a Chernoff bound, we bound the error probability by

$$\mathbb{P}(I < 12 \log(1/\delta)) = \mathbb{P}(I < (1 - 1/3)\,\mathbb{E}(I)) \leq \exp(-18 \log(1/\delta)/(2 \cdot 9)) = \delta.$$

Now, consider the case when $\sum_i E_i \leq k$. Let $g := K/(64 \cdot k \log K)$ be the gap which the recursive call to ALGBOOSTED distinguishes. Define $\tilde{L}_\ell := \{\, j \mid E_j \geq 2^\ell / g \,\}$. Note that for all ℓ, it holds that $|\tilde{L}_\ell| \le K/(64 \cdot 2^\ell \log K)$. Indeed, if there was some ℓ with $|\tilde{L}_\ell| > K/(64 \cdot 2^\ell \log K)$ then

$$k \ge \sum_{i=1}^{s} E_i \ge |\tilde{L}_\ell| \frac{2^\ell}{g} > \frac{K}{64 g \log K} = k;$$

a contradiction. Focus on iteration $d = 2^\ell$ of the for-loop in Line 16. For every $i \in \tilde{L}_\ell$, define the indicator random variable I_i which equals one if $i \in S$. Let $I := \sum_{i \in \tilde{L}_\ell} I_i$. Since we sample S with rate $\frac{108d}{K} \log K \log(1/\delta)$, it follows that

$$\mathbb{E}(I) = |\tilde{L}_\ell| \cdot \frac{108d}{K} \log K \log(1/\delta) \le \frac{K}{64d \log K} \cdot \frac{108d}{K} \log K \log(1/\delta) \le 1.7 \log(1/\delta).$$

Note that the set $\tilde{L}_\ell$ contains all subproblems E_i for which AlgBoosted may return Far. Thus, the algorithm correctly returns Close if for every iteration $I < 12 \log(1/\delta)$ (see Line 19). By a Chernoff bound we can bound this error probability as

$$\mathbb{P}(I \ge 12 \log(1/\delta)) \le \mathbb{P}(I \ge 7\,\mathbb{E}(I)) \le 2^{-7\,\mathbb{E}(I)} \le \exp(-\log(1/\delta)) = \delta.$$

The claim follows by taking a union bound over the $\log K$ iterations and recalling our choice of $\delta = 0.01/\log K$. ◁

Finally, note that the lemma statement follows by the inductive hypothesis (i), that is, by calling AlgBoosted with error probability $\delta := 1/3$. □

6.3 Main Theorem

We now put things together to prove Main Theorem 4.3 and its corollaries.

Main Theorem 4.3. *Let $2 \le \Delta \le n$ be a parameter. Then, there is a randomized algorithm that solves the (k, K)-gap edit distance problem in time $O(n/K + kK) \cdot \Delta^3 \cdot (\log n)^{O(\log_\Delta n)}$ and succeeds with constant probability, provided that $K/k \ge (\log n)^{c \cdot \log_\Delta(n)}$ for a sufficiently large constant $c > 0$.*

Proof. We run Lemma 6.4 with parameters $X, Y, k, K, \Delta' := \max(\Delta, (256 \log K)^2)$ and $p := n$. Note that $\mathrm{BP}_p(X) = 1 \le p \cdot K/k$ holds, so the call is valid. If $\Delta \ge (256 \log K)^2$, the lemma follows immediately by the guarantees of Lemma 6.4. If $2 \le \Delta < (256 \log K)^2$, then observe that by setting $\Delta' = (256 \log K)^2$ the gap and the running time of Lemma 6.4 do not increase compared to the lemma statement, which completes the proof. □

Corollary 4.4 (Subpolynomial Gap)**.** *The $(k, k \cdot 2^{\Theta(\sqrt{\log k \log\log k})})$-gap edit distance problem is in time $O(n/k + k^{2+o(1)})$.*

Proof. If $k^6 < n$, run the algorithm of [Bri+22b, Corollary 2] to solve the problem directly in time $O(n/k)$. So we can assume that $n \leq k^6$. In particular, $\log n = \Theta(\log k)$. We use Main Theorem 4.3 setting $\Delta = 2^{\sqrt{\log k \log\log k}}$. This gives an algorithm for the (k, K)-gap problem running in time $O\left((n/K + kK) \cdot 2^{c_0 \cdot \sqrt{\log k \log\log k)}}\right)$ for some constant $c_0 > 0$, where $K/k \geq 2^{c\sqrt{\log k \log\log k}}$ with $c > 0$ sufficiently large. Thus, setting $K = k \cdot 2^{\mu \cdot \sqrt{\log k \log\log k}}$ where $\mu > c_0 + c$ yields the result. □

Corollary 4.5 (Polylogarithmic Gap)**.** *For any constant $\varepsilon > 0$, the $(k, k \cdot (\log k)^{\Theta(1/\varepsilon)})$-gap edit distance problem is in time $O(n/k^{1-\varepsilon} + k^{2+\varepsilon})$.*

Proof. If $k^6 < n$, we solve the problem directly in time $O(n/k^{1-\varepsilon})$ using the algorithm of [Bri+22b, Corollary 3]. Thus, we can assume from now on that $n \leq k^6$ and therefore $\log n = \Theta(\log k)$. We use Main Theorem 4.3 choosing $\Delta := k^{\delta/3}$ for some $0 < \delta < \varepsilon$, and $K = k(\log k)^{O(1/\varepsilon)}$ with large enough hidden constant. This yields an algorithm for the desired gap running in time

$$(n/k + k^2) \cdot k^{\delta} \cdot (\log k)^{O(1/\delta)} \leq O(n/k^{1-\varepsilon} + k^{2+\varepsilon}).$$ □

Corollary 4.6 (Polynomial Gap)**.** *Let k, K be such that $K > k^{1+\varepsilon}$ for some constant $\varepsilon > 0$. Then the (k, K)-gap edit distance problem is in time $\widehat{O}(n/K + \sqrt{nk} + k^2)$.*

Proof. If $K < 2^{(\log n)^{2/3}}$, then we can solve the (k, K)-gap edit distance problem exactly using the Landau Vishkin algorithm in time $O(n + k^2) \subseteq O(n^{1+o(1)}/K + k^2)$. Hence, since $K > k^{1+\varepsilon}$ we can assume from now on that $K/k \geq 2^{\Theta((\log n)^{2/3})}$.

Instantiating Main Theorem 4.3 with $\Delta = 2^{\sqrt{\log n \log\log n}}$, we obtain an algorithm that can distinguish the gap $\tilde{K}/k \geq 2^{\Theta(\sqrt{\log n \log\log n})}$ in time $O((n/\tilde{K} + k\tilde{K}) \cdot n^{o(1)})$. If $K < \sqrt{n/k}$ then running this algorithm with $\tilde{K} := K$ solves the (k, K)-gap problem in time $O(n^{1+o(1)}/K)$. (Here we used the assumption that $K/k \geq 2^{\Theta((\log n)^{2/3})}$, since this is larger than the minimum gap distinguishable by Main Theorem 4.3.) Otherwise, note that running this algorithm with $\tilde{K} := \max(\sqrt{n/k}, k \cdot 2^{\Theta(\sqrt{\log n \log\log n})})$ correctly solves the (k, K)-gap problem, since we do not increase the gap. In this case, the running time is $O((\sqrt{nk} + k^2) \cdot n^{o(1)})$. Combining the above yields the claimed running time. □

Part III

Negative Single Source Shortest Paths

7 Introduction to SSSP

This part of the thesis is based on our publication [BCF23]. I contributed an equal share of the work, and at least one third of the write-up.

[BCF23]	Karl Bringmann, Alejandro Cassis, and Nick Fischer. “Negative-Weight Single-Source Shortest Paths in Near-Linear Time: Now Faster!” In: *FOCS*. IEEE, 2023, pp. 515–538. DOI: 10.1109/FOCS57990.2023.00038.

In this part of the thesis, we study the Single-Source Shortest Paths (SSSP) problem with possibly negative integer edge weights: Given a directed weighted graph G and a designated source vertex s, compute the distances from s to all other vertices in G. This is possibly *the* most fundamental weighted graph problem with wide-ranging applications in computer science, including routing, data networks, artificial intelligence, planning, and operations research.

While it is well known for almost 60 years that SSSP with nonnegative edge weights can be solved in near-linear time by Dijkstra’s algorithm [Dij59; Wil64; FT87], the case with negative weights has a more intriguing history: The Bellman-Ford algorithm was developed in the ’50s [Shi55; For56; Bel58; Moo59] and runs in time $O(mn)$, and this time bound remained the state of the art for a long time. Starting in the ’80s, the *scaling technique* was developed and lead to time $O(m\sqrt{n}\log W)$ [Gab83; GT89; Gol95]; here and throughout, W is the magnitude of the smallest negative edge weight in the graph.[1] Other papers focused on specialized graph classes, leading e.g. to near-linear time algorithms for planar directed graphs [LRT79; Hen+97; FR06; KMW09], and improved algorithms for dense graphs with small weights [San05].

An alternative approach is to model SSSP as a minimum-cost flow problem.[2] In the last decade, an impressive combination of convex optimization techniques and dynamic algorithms have resulted in a series of advancements in minimum-cost flow computations [Coh+17; AMV20; Bra+20; Bra+21] and thus also for negative-weight SSSP, with running times $\widetilde{O}(m^{10/7})$ [Coh+17], $\widetilde{O}(m^{4/3})$ [AMV20] and $\widetilde{O}(m + n^{3/2})$ [Bra+20].

1. Strictly speaking, $W \geq 0$ is the smallest integer such that all edge weights satisfy $w(e) \geq -W$. By slight abuse of notation we write $O(\log W)$ to express $O(\max\{1, \log W\})$.
2. To model SSSP as a minimum-cost flow problem, interpret each edge e with weight $w(e)$ as an edge with infinite capacity and cost $w(e)$. Moreover, add an artificial sink vertex t to the graph, and add unit-capacity cost-0 edges from all vertices v to t. Then any minimum-cost flow routing n units from s to t corresponds exactly to a shortest path tree in the original graph (assuming that it does not contain a negative-weight cycle).

This line of research recently culminated in a randomized almost-linear $m^{1+o(1)}$-time algorithm, by Chen, Kyng, Liu, Peng, Probst Gutenberg and Sachdeva [Che+22] (very recently derandomized by van den Brand, Chen, Kyng, Liu, Peng, Probst Gutenberg, Sachdeva and Sidford [Bra+23]).

Finally, at the same time as the breakthrough in computing minimum-cost flows, Bernstein, Nanongkai and Wulff-Nilsen [BNW22] found an astonishing[3] *near-linear* $O(m \log^8(n) \log(W))$-time algorithm for negative-weight SSSP. We will refer to their algorithm as the *BNW algorithm*. The BNW algorithm is combinatorial and arguably simple (certainly simpler than the minimum-cost flow algorithms), and thus a satisfying answer to the coarse-grained complexity of the negative-weight SSSP problem. However, the story does not end here. In this work, we press further and investigate the following question which was left open by Bernstein, Nanongkai and Wulff-Nilson [BNW22]:

Can we further improve the number of log-factors
in the running time of negative-weight SSSP?

For comparison, the *nonnegative*-weights SSSP problem underwent a long series of lower-order improvements in the last century [Dij59; FT87; FW93; FW94; Tho00; Ram96; Ram97; Emd77; EKZ77; MN90; Ahu+90; CGS99; Tho04], including improvements by log-factors or even just loglog-factors.[4] In the same spirit, we initiate the fine-grained study of lower-order factors for negative-weight shortest paths.

7.1 Our Results

In our main result we make significant progress on our driving question, and improve the BNW algorithm by nearly six log-factors:

Main Theorem 7.1 (Negative-Weight SSSP)**.** *There is a Las Vegas algorithm which, given a directed graph G and a source node s, either computes a shortest path tree from s or finds a negative cycle in G, running in time $O((m + n \log\log n) \log^2 n \log(nW))$ with high probability (and in expectation).*

We obtain this result by optimizing the BNW algorithm, pushing it to its limits. Aaron Bernstein remarked in a presentation of their result that "something like $\log^5$ is inherent to [their] current framework".[5] It is thus surprising that we obtain such a dramatic improvement to nearly three log-factors within the same broader framework. Despite this speed-up, our algorithm is still modular and simple in its core. In Section 7.2 we discuss the technical similarities and differences between our algorithm and the BNW algorithm in detail.

3. If the reader is reluctant to our choice of adjectives to describe these results, we refer them to the recent CACM article "Historic Algorithms Help Unlock Shortest-Path Problem Breakthrough" [Edw23].
4. In these papers, the Dijkstra running time $O(m + n \log n)$ was improved to the current state of the art $O(m + n \log\log \min\{n, C\})$ [Tho04], where C is the largest weight in the graph.
5. `https://youtu.be/Bpw3yqWT_d0?t=3721`

Recall that computing shortest paths is only reasonable in graphs without negative cycles (as otherwise two nodes are possibly connected by a path of arbitrarily small negative weight). In light of this, we solve the negative-weight SSSP problem in its strongest possible form in Main Theorem 7.1: The algorithm either returns a shortest path tree, or returns a negative cycle as a certificate that no shortest path tree exists. In fact, the subproblem of detecting negative cycles has received considerable attention on its own in the literature (see e.g. the survey [CG99]).

In the presence of negative cycles, a natural alternative to finding one such cycle is to compute all distances in the graph anyway (where some of the distances are $-\infty$ or ∞). This task can be solved in the same running time:

Theorem 7.2 (Negative-Weight Single-Source Distances)**.** *There is a Las Vegas algorithm, which, given a directed graph G and a source $s \in V(G)$, computes the distances from s to all other vertices in the graph (these distances are possibly $-\infty$ or ∞), running in time $O((m + n \log\log n) \log^2 n \log(nW))$ with high probability (and in expectation).*

Owing to the countless practical applications of shortest paths problems, it is an important question to ask whether there is a negative-weights shortest paths algorithm that has a competitive implementation. The typical practical implementation in competitive programming uses optimized variants of Bellman-Ford's algorithm, such as the "Shortest Path Faster Algorithm" [Moo59; Con23] (see also [Che+08] for an experimental evaluation of other more sophisticated variants of Bellman-Ford). However, it is easy to find instances for which these algorithms require time $\Omega(mn)$. It would be exciting if, after decades of competitive programming, there finally was an efficient implementation to deal with these instances. With its nine log-factors, the BNW algorithm does not qualify as a practical candidate. We believe that our work paves the way for a comparably fast implementation.

In addition to our main result, we make progress on two closely related problems: Computing the minimum cycle mean, and low-diameter decompositions in directed graphs. We describe these results in the following sections.

Minimum Cycle Mean

In a directed weighted graph, the *mean* of a cycle C is defined as the ratio $\bar{w}(C) = w(C)/|C|$ where $w(C)$ is the total weight of C. The *Minimum Cycle Mean* problem is to compute, in a given directed weighted graph, the minimum mean across all cycles, $\min_C \bar{w}(C)$. This is a central problem in the context of network flows [AMO93], with applications to verification and reactive systems analysis [Cha+14].

There is a large body of literature on computing the Minimum Cycle Mean. In 1987, Karp [Kar78] established an $O(mn)$-time algorithm, which is the fastest strongly polynomial time algorithm to this date. In terms of weakly polynomial algorithms, Lawler observed that the problem is equivalent to detecting negative cycles, up to a factor $O(\log(nW))$ [Law66; Law76]. Indeed, note that one direction is trivial: The graph has a negative cycle if and only if the minimum cycle mean is negative. For the other direction, he provided a reduction to detecting negative cycles on $O(\log(nW))$ graphs with modified rational edge weights. Thus, following Lawler's observation, any negative-weight SSSP algorithm can be turned into a Minimum Cycle Mean algorithm in a black-box way with running time overhead $O(\log(nW))$.

There are also results specific to Minimum Cycle Mean computations: Orlin and Ahuja [OA92] designed an algorithm in time $O(m\sqrt{n}\log(nW))$ (improving over the baseline $O(m\sqrt{n}\log^2(nW))$ which follows from the SSSP algorithms by [Gab83; GT89; Gol95]). For the special case of dense graphs with 0-1-weights, an $O(n^2)$-time algorithm is known [BC92]. Finally, in terms of approximation algorithms it is known how to compute a $(1+\varepsilon)$-approximation in time $\widetilde{O}(n^\omega \log(W)/\varepsilon)$ [Cha+14].

As for negative-weight SSSP, all these algorithms are dominated by the recent BNW algorithm: By Lawler's observation, their algorithm computes the minimum cycle mean in time $O(m\log^8(n)\log^2(nW))$. In fact, it is implicit in their work that the running time can be reduced to $O(m\log^8(n)\log(nW))$. Our contribution is again that we reduce the number of log-factors from nine to nearly three:

Theorem 7.3 (Minimum Cycle Mean)**.** *There is a Las Vegas algorithm, which given a directed graph G finds a cycle C with minimum mean weight $\bar{w}(C) = \min_{C'} \bar{w}(C')$, running in time $O((m + n\log\log n)\log^2 n \log(nW))$ with high probability (and in expectation).*

Directed Low-Diameter Decompositions

A crucial ingredient to the BNW algorithm is a Low-Diameter Decomposition (LDD) in directed graphs. Our SSSP algorithm differs in that regard from the BNW algorithm, and does not explicitly use LDDs. Nevertheless, as a side result of this work we improve the best known LDD in directed graphs.

LDDs have been first studied by Awerbuch almost 40 years ago [Awe85] and have ever since found several applications, mostly for undirected graphs and mostly in distributed, parallel and dynamic settings [Awe+89; AP92; Awe+92; LS93; Bar96; Ble+14; MPX13; Pac+18; FG19; CZ20; BGW20; FGV21; BNW22]. The precise definitions in these works mostly differ, but the common idea is to select a small subset of edges S such that after removing all edges in S from the graph, the remaining graph has (strongly) connected components with bounded diameter.

For directed graphs, we distinguish two types of LDDs: *Weak* LDDs ensure that for every strongly connected component C in the graph $G \setminus S$, the diameter of C *in the original graph* is bounded. A *strong* LDD exhibits the stronger property that the diameter of C in the graph $G \setminus S$ is bounded.

Definition 7.4 (Directed Low-Diameter Decomposition)**.** *A* weak Low-Diameter Decomposition *with overhead ρ is a Las Vegas algorithm that, given a directed graph G with nonnegative edge weights w and a parameter $D > 0$, computes an edge set $S \subseteq E(G)$ with the following properties:*

- Sparse Hitting: *For any edge $e \in E$, $\mathbb{P}(e \in S) \leq O(\frac{w(e)}{D} \cdot \rho + \frac{1}{\mathrm{poly}(n)})$.*
- Weak Diameter: *Every SCC C in $G \setminus S$ has weak diameter at most D (that is, for any two vertices $u, v \in C$, we have $\mathrm{dist}_G(u, v) \leq D$).*

We say that the Low-Diameter Decomposition is strong *if it additionally satisfies the following stronger property:*

- Strong Diameter: *Every SCC C in $G \setminus S$ has diameter at most D (that is, for any two vertices $u, v \in C$, we have $\mathrm{dist}_{G \setminus S}(u, v) \leq D$).*

For directed graphs, the state-of-the-art *weak* LDD was developed by Bernstein, Nanongkai and Wulff-Nilsen [BNW22] as a tool for their shortest paths algorithm. Their result is a weak LDD with polylogarithmic overhead $O(\log^2 n)$ running in near-linear time $O(m \log^2 n + n \log^2 n \log \log n)$. In terms of *strong* LDDs, no comparable result is known. While it is not hard to adapt their algorithm to compute a strong LDD, this augmentation suffers from a slower running time $\Omega(nm)$. Our contribution is designing the first strong LDD computable in near-linear time, with only slightly worse overhead $O(\log^3 n)$:

Theorem 7.5 (Strong Low-Diameter Decomposition)**.** *There is a strong Low-Diameter Decomposition with overhead $O(\log^3 n)$, computable in time $O((m + n \log \log n) \log^2 n)$ with high probability (and in expectation).*

7.2 Technical Overview

Our algorithm is inspired by BNW algorithm and follows its general framework, but differs in many aspects. In this section we give a detailed comparison.

The Framework

The presentation of our algorithm is modular: We will first focus on the SSSP problem on a restricted class of graphs (to which we will simply refer as *restricted* graphs, see the next Definition 7.6). In the second step we demonstrate how to obtain our results for SSSP on general graphs, for finding negative cycles, and for computing the minimum cycle mean by reducing to the restricted problem in a black-box manner.

Definition 7.6 (Restricted Graphs)**.** *An edge-weighted directed graph $G = (V, E, w)$ with a designated source vertex $s \in V$ is* restricted *if it satisfies:*

- *The edge weights are integral and at least -1.*
- *The minimum cycle mean is at least 1.*
- *The source s is connected to every other vertex by an edge of weight 0.*

In particular, note that restricted graphs do not contain negative cycles, and therefore it is always possible to compute a shortest path tree. The *Restricted SSSP* problem is to compute a shortest path tree in a given restricted graph G. We write $T_{\mathrm{RSSSP}}(m, n)$ for the optimal running time of a Restricted SSSP algorithm with error probability $\frac{1}{2}$, say.

Improvement 1: Faster Restricted SSSP via Better Decompositions

Bernstein et al. [BNW22] proved that $T_{\mathrm{RSSSP}}(m, n) = O(m \log^5 n)$. Our first contribution is that we shave nearly three log-factors and improve this bound to $T_{\mathrm{RSSSP}}(m, n) = O((m + n \log\log n) \log^2 n)$ (see Theorem 8.1).

At a high level, the idea of the BNW algorithm is to decompose the graph by finding a subset of edges S suitable for the following two subtasks: (1) We can recursively compute shortest paths in the graph $G \setminus S$ obtained by removing the edges in S, and thereby make enough progress to incur in total only a small polylogarithmic overhead in the running time. And (2), given the outcome of the recursive call, we can efficiently "add back" the edges from S to obtain a correct shortest path tree for G. For the latter task, the crucial property is that S intersects every shortest path in G at most $O(\log n)$ times (in expectation), as then a simple generalization of Dijkstra's and Bellman-Ford's algorithm can adjust the shortest path tree in near-linear time (see Lemma 8.8).

For our result, we keep the implementation of step (2) mostly intact, except that we use a faster implementation of Dijkstra's algorithm due to Thorup [Tho04] (see Lemma 8.8). The most significant difference takes place in step (1), where we change how the algorithm selects S. Specifically, Bernstein et al. used a directed Low-Diameter Decomposition to implement the decomposition. We are following the same thought, but derive a more efficient and direct decomposition scheme. To this end, we define the following key parameter:

Definition 7.7. *Let G be a restricted graph with designated source s. We define $\kappa(G)$ as the maximum number of negative edges (that is, edges of weight exactly -1) in any simple path P which starts at s and has nonpositive weight $w(P) \leq 0$.*

Our new decomposition can be stated as follows.

Lemma 7.8 (Decomposition)**.** *Let G be a restricted graph with source vertex $s \in V(G)$ and $\kappa \geq \kappa(G)$. There is a randomized algorithm* Decompose(G, κ) *running in expected time $O((m + n \log\log n) \log n)$ that computes an edge set $S \subseteq E(G)$ such that:*

1. Progress: *With high probability, for any strongly connected component C in $G \setminus S$, we have (i) $|C| \leq \frac{3}{4}|V(G)|$ or (ii) $\kappa(G[C \cup \{ s \}]) \leq \frac{\kappa}{2}$.*
2. Sparse Hitting: *For any shortest s-v-path P in G, we have $\mathbb{E}(|P \cap S|) \leq O(\log n)$.*

The sparse hitting property is exactly what we need for (2). With the progress condition, we ensure that $|V(G)| \cdot \kappa(G)$ reduces by a constant factor when recurring on the strongly connected components of $G \setminus S$. The recursion tree therefore reaches depth at most $O(\log(n \cdot \kappa(G))) = O(\log n)$. In summary, with this new idea we can compute shortest paths in restricted graphs in time $O((m + n \log\log n) \log^2 n)$.

Improvement 2: Faster Scaling

It remains to lift our Restricted SSSP algorithm to the general SSSP problem at the expense of at most one more log-factor $\log(nW)$. In comparison, the BNW algorithm spends four log-factors $O(\log^3 n \log W)$ here. As a warm-up, we assume that the given graph is promised not to contain a negative cycle.

Warm-Up: From Restricted Graphs to Graphs without Negative Cycles This task is a prime example amenable to the *scaling technique* from the '80s [Gab83; GT89; Gol95]: By rounding the weights in the given graph G from $w(e)$ to $\lceil \frac{3w(e)}{W+1} \rceil + 1$ we ensure that (i) all weights are at least -1 and (ii) the minimum cycle mean is at least 1, and thus we turn G into a restricted graph H (see Lemma 8.20). We compute the shortest paths in H and use the computed distances (by means of a *potential function*) to augment the weights in the original graph G. If G has smallest weight $-W$, in this way we can obtain an *equivalent* graph G' with smallest weight $-\frac{3}{4}W$, where equivalence is defined as follows:

Definition 7.9 (Equivalent Graphs)**.** *We say that two graphs G, G' over the same set of vertices and edges are* equivalent *if (1) any shortest path in G is also a shortest path in G' and vice versa, and (2) for any cycle C, $w_G(C) = w_{G'}(C)$.*

Hence, by (1) we continue to compute shortest paths in G'. At first glance it seems that repeating this scaling step incurs only a factor $\log W$ to the running time, but for subtle reasons the overhead is actually $\log(nW)$. Another issue is that the Restricted SSSP algorithm errs with constant probability. The easy fix loses another $\log n$ factor due to boosting (this is how Bernstein et al. obtain their algorithm SPMonteCarlo, see [BNW22, Theorem 7.1]). Fortunately, we can "merge" the scaling and boosting steps to reduce the overhead to $\log(nW)$ in total, see Theorem 8.19.

From Restricted Graphs to Arbitrary Graphs What if G contains a negative cycle? In this case, our goal is to find and return one such negative cycle. Besides the obvious advantage that it makes the output more informative, this also allows us to strengthen the algorithm from Monte Carlo to Las Vegas, since both a shortest path tree and a negative cycle serve as *certificates* that can be efficiently tested. Using the scaling technique as before, we can easily *detect* whether a given graph contains a negative cycle in time $O(T_{\mathrm{RSSSP}}(m, n) \cdot \log(nW))$ (even with high probability, see Corollary 8.24), but we cannot *find* such a cycle.

We give an efficient reduction from finding negative cycles to Restricted SSSP with overhead $O(\log(nW))$. This reduction is the technically most involved part of our work. In the following paragraphs we attempt to give a glimpse into the main ideas.

A Noisy-Binary-Search-Style Problem For the rest of this overview, we phrase our core challenge as abstract as possible, and omit further context. We will use the following notation: given a directed graph G and an integer M, we write G^{+M} to denote the graph obtained by adding M to every edge weight of G. Consider the following task:

Definition 7.10 (Threshold)**.** *Given a weighted graph G, compute the smallest integer $M^* \geq 0$ such that the graph G^{+M^*}, which is obtained from G by adding M^* to all edge weights, does not contain a negative cycle.*

Our goal is to solve the Threshold problem in time $O(T_{\text{RSSSP}}(m,n)\log(nW))$ (from this it follows that we can find negative cycles in the same time, see Lemma 8.26). As a tool, we are allowed to use the following lemma as a black-box (which can be proven similarly to the warm-up case):

Lemma 7.11 (Informal Lemma 8.30)**.** *There is an $O(T_{RSSSP}(m,n))$-time algorithm that, given a graph G with minimum weight $-W$, either returns an equivalent graph G' with minimum weight $-\frac{3}{4}W$, or returns* NegativeCycle*. If G does not contain a negative cycle, then the algorithm returns* NegativeCycle *with error probability at most* 0.01.

Morally, Lemma 7.11 provides a test whether a given graph G contains a negative cycle. A natural idea is therefore to find M^* by binary search, using Lemma 7.11 as the tester. However, note that this tester is *one-sided*: If G contains a negative cycle, then the tester is not obliged to detect one. Fortunately, we can turn the tester into a win-win algorithm to compute M^*.

We first describe our Threshold algorithm in an idealized setting where we assume that the tester from Lemma 7.11 has error probability 0. We let $d = \frac{1}{5}W$, and run the tester on the graph G^{+d}. There are two cases:

- *The tester returns* NegativeCycle*:* In the idealized setting we can assume that G^{+d} indeed contains a negative cycle. We therefore compute the threshold of G^{+d} recursively, and return that value plus d. Note that the minimum weight of G^{+d} is at least $-W + d = -\frac{4}{5}W$.

- *The tester returns an equivalent graph G':* In this case, we recursively compute and return the threshold value of $(G')^{-d}$. Note that the graphs G and $(G')^{-d}$ share the same threshold value, as by Definition 7.9 we have $w_{(G')^{-d}}(C) = w_{G'}(C) - d = w_{G^{+d}}(C) - d = w_G(C)$ for any cycle C. Moreover, since G^{+d} has smallest weight $-\frac{4}{5}W$, the equivalent graph G' has smallest weight at least $-\frac{3}{4} \cdot \frac{4}{5}W = -\frac{3}{5}W$ by Lemma 7.11. Therefore, $(G')^{-d}$ has smallest weight at least $-\frac{3}{5}W - d = -\frac{4}{5}W$.

In both cases, we recursively compute the threshold of a graph with smallest weight at least $-\frac{4}{5}W$. Therefore, the recursion reaches depth $O(\log W)$ until we have reduced the graph to constant minimum weight and the problem becomes easy.

The above algorithm works in the idealized setting, but what about the unidealized setting, where the tester can err with constant probability? We could of course first boost the tester to succeed with high probability. In combination with the above algorithm, this would solve the Threshold problem in time $O(T_{\text{RSSSP}}(m, n) \log(nW) \log n)$.

However, the true complexity of this task lies in avoiding the naive boosting. By precisely understanding the unidealized setting with constant error probability, we improve the running time for Threshold to $O(T_{\text{RSSSP}}(m, n) \log(nW))$. To this end, it seems that one could apply the technique of *noisy binary search* (see e.g. [Pel89; Fei+94; Pel02]). Unfortunately, the known results do not seem applicable to our situation, as Lemma 7.11 only provides a one-sided tester. Our solution to this final challenge is an innovative combination of the algorithm sketched above with ideas from noisy binary search. The analysis makes use of *drift analysis* (see e.g. [Len17]), which involves defining a suitable *drift function* (a quantity which in expectation decreases by a constant factor in each step and is zero if and only if we found the optimal value M^*) and an application of a *drift theorem* (see Theorem 8.34) to prove that the drift function rapidly approaches zero.

7.3 Summary of Log Shaves

Finally, to ease the comparison with the BNW algorithm, we compactly list where exactly we shave the nearly six log-factors. We start with the improvements in the Restricted SSSP algorithm:

- We use Thorup's priority queue [Tho04] to speed up Dijkstra's algorithm in Lemma 8.8. This reduces the cost of one log-factor to a loglog-factor.

- The Sparse Hitting property of our decomposition scheme (Lemma 7.8) incurs only an $O(\log n)$ overhead in comparison to the $O(\log^2 n)$ overhead due to the Low-Diameter Decomposition in the BNW algorithm.

- The Progress property of our decomposition scheme (Lemma 7.8) ensures that the recursion depth of our Restricted SSSP algorithm is just $O(\log n)$. The analogous recursion depth in the BNW algorithm is $O(\log^2 n)$ (depth $\log n$ for reducing the number of nodes n times depth $\log n$ for reducing $\max_v \eta_G(v)$).

Next, we summarize the log-factors shaved in the scaling step:

- The BNW algorithm amplifies the success probability of the Restricted SSSP algorithm by repeating it $O(\log n)$ times. We combine the boosting with the scaling steps which saves this log-factor.

- We improve the overall reduction from finding a negative cycle to Restricted SSSP. In particular, we give an implementation of Threshold which is faster by two log-factors (see Lemmas 8.26 and 8.32). This is where we use an involved analysis via a drift function.

7.4 Open Problems

We leave several interesting open questions:

1. *Can the number of* $\log n$ *factors be improved further?*
   In our algorithm, we suffer three log-factors because of (i) the scaling technique ($\log(nW)$) to reduce to restricted graphs, and on restricted graphs (ii) the inherent $\log n$ overhead of the graph decomposition and (iii) the recursion depth $\log n$ to progressively reduce $\kappa(G)$, all of which seem unavoidable. We therefore believe that it is hard to improve upon our algorithm without substantially changing the framework.

2. *Can the loss due to the scaling technique be reduced from* $\log(nW)$ *to* $\log W$*?*
   The classical scaling technique, as a reduction to graphs with weights at least -1, requires only $\log W$ iterations [Gol95]. But in our setting, due to the stronger conditions for *restricted* graphs (and due to the boosting), we need $\log(nW)$ iterations. Can we do better?

3. *Can the* $\log W$ *factor be removed from the running time altogether?*
   That is, is there a *strongly polynomial* algorithm in near-linear time? In terms of strongly polynomial algorithms, perhaps surprisingly, the Bellman-Ford algorithm with running time $O(nm)$ has remained the state of the art for a long time. Only very recently, a preprint by Fineman [Fin23] claims an algorithm in time $\widetilde{O}(mn^{8/9})$, breaking through this barrier.[6]

4. *Can the algorithm be derandomized?*
   The fastest deterministic algorithm for negative-weight SSSP is obtained via the recent almost-linear $O(m^{1+o(1)} \log W)$-time deterministic minimum-cost flow algorithm by van den Brand, Chen, Kyng, Liu, Peng, Probst Gutenberg, Sachdeva and Sidford [Bra+23]. Can we obtain a deterministic near-linear time algorithm for negative-weight SSSP? The current barrier to derandomize our approach (and BNW's) is that the sparse hitting property of Lemma 7.8 is inherently probabilistic.

5. *Simpler and practical algorithms*
   The recent breakthroughs showing that minimum cost flow is in almost linear time [Che+22; Bra+23], settled the *right time complexity* of various fundamental graph problems like maximum bipartite matching, optimal transport, edge/vertex connectivity, etc. However, the current techniques used by these works are *very far* from being practical or to any notion of "simple". We believe that trying to obtain *simple and practical* algorithms in almost linear time for any of these problems is a very interesting and important research direction.

6. In fact, Fineman considers the more challenging setting where the edge weights might be real numbers (in a reasonable machine model). Any scaling-based approach inherently breaks to solve this variant. See another related recent result by Karczmarz, Nadara and Sokolowski [KNS24] where they study SSSP with rational weights and improve upon Bellman-Ford in this setting.

Our work (building upon the BNW algorithm [BNW22]) can be seen as a step in that direction: Besides the theoretical importance of improving the running time of negative-weight SSSP, a driving research question for us has been trying to obtain a *practical* algorithm for this fundamental problem. We believe that our work paves the way for such an implementation. Another very recent example that goes in that direction is an improved "combinatorial" algorithm for maximum bipartite matching by Chuzhoy and Khanna [CK24].

7.5 Outline

The remaining of this part of the thesis is structured as follows. In Section 7.6 we give some formal preliminaries. Chapter 8 is devoted to present our algorithm for negative-weight SSSP, proving Main Theorem 7.1. In Chapter 9 we present our side results: in Section 9.1 we give our result for computing the minimum cycle mean (Theorem 7.3), and in Section 9.2 we give our strong Low-Diameter Decomposition for directed graphs (Theorem 7.5).

7.6 Preliminaries

We write $[n] = \{ 1, \dots, n \}$ and $\widetilde{O}(T) = T \cdot (\log T)^{O(1)}$. We say that an event occurs *with high probability* if it occurs with probability $1 - 1/n^c$ for an arbitrarily large constant c (here, n is the number of vertices in the input graph). Unless further specified, our algorithms are Monte Carlo algorithms that succeed with high probability.

Directed Graphs We work with directed edge-weighted graphs $G = (V, E, w)$. Here, $V = V(G)$ is the set of vertices and $E = E(G) \subseteq V(G)^2$ is the set of edges. All edge weights are integers, denoted by $w(e) = w(u, v)$ for $e = (u, v) \in E(G)$. We typically set $n = |V(G)|$ and $m = |E(G)|$. We write $G[C]$ to denote the *induced subgraph* with vertices $C \subseteq V(G)$ and write $G \setminus S$ to denote the graph G after deleting all edges in $S \subseteq E$. We write $\deg(v)$ for the (out-)degree of v, that is, the number of edges starting from v.

A *strongly connected component (SCC)* is a maximal set of vertices $C \subseteq V(G)$ in which all pairs are reachable from each other. It is known that every directed graph can be decomposed into a collection of SCCs, and the graph obtained by compressing the SCCs into single nodes is acyclic. It is also known that the SCCs can be computed in linear time:

Lemma 7.12 (Strongly Connected Components, [Tar72]). *In any directed graph G, the strongly connected components can be identified in time $O(n + m)$.*

For a set of edges S (such as a path or a cycle), we write $w(S) = \sum_{e \in S} w(e)$. A negative cycle is a cycle C with $w(C) < 0$. For vertices u, v, we write $\mathrm{dist}_G(u, v)$ for the length of the shortest u-v-path. If there is a negative-weight cycle in some u-v-path, we set $\mathrm{dist}_G(u,v) = -\infty$, and if there is no path from u to v we set $\mathrm{dist}_G(u,v) = \infty$.

Definition 7.13 (Balls). *For a vertex v, and a nonnegative integer r, we denote the out-ball centered at v with radius r by $B^{out}_G(v,r) = \{\, u \in V(G) : \mathrm{dist}_G(v,u) \le r \,\}$. Similarly, we denote the in-ball centered at v with radius r by $B^{in}_G(v,r) = \{\, u \in V(G) : \mathrm{dist}_G(u,v) \le r \,\}$. Further, we write $\partial B^{out}_G(v,r) = \{\, (u,w) \in E : u \in B^{out}(v,r) \wedge w \notin B^{out}(v,r) \,\}$ to denote the boundary edges of an out-ball, and $\partial B^{in}_G(v,r) = \{\, (u,w) \in E : u \notin B^{in}(v,r) \wedge w \in B^{in}(v,r) \,\}$ for an in-ball.*

In all these notations, we occasionally drop the subscript G if it is clear from context.

The *Single-Source Shortest Paths (SSSP)* problem is to compute the distances $\mathrm{dist}_G(s,v)$ for a designated source vertex $s \in V(G)$ to all other vertices $v \in V(G)$. When G does not contain negative cycles, this is equivalent to compute a *shortest path tree* from s (that is, a tree in which every s-to-v path is a shortest path in G). For graphs with *nonnegative* edge weights, Dijkstra's classical algorithm solves the SSSP problem in near-linear time. We use the following result by Thorup, which replaces the $\log n$ overhead by $\log\log n$ (in the RAM model, see the paragraph on the machine model below).

Lemma 7.14 (Dijkstra's Algorithm, [Dij59; Tho04]). *In any directed graph G with nonnegative edge weights, the SSSP problem can be solved in time $O(m + n \log\log n)$.*

Lemma 7.15 (Bellman-Ford's Algorithm, [Shi55; For56; Bel58; Moo59]). *In any directed graph G, the SSSP problem can be solved in time $O(mn)$.*

Potentials Let G be a directed graph. We refer to functions $\phi : V(G) \to \mathbb{Z}$ as *potential functions*. We write G_ϕ for the graph obtained from G by changing the edge weights to $w_\phi(u,v) = w(u,v) + \phi(u) - \phi(v)$.

Definition 7.9 (Equivalent Graphs). *We say that two graphs G, G' over the same set of vertices and edges are* equivalent *if (1) any shortest path in G is also a shortest path in G' and vice versa, and (2) for any cycle C, $w_G(C) = w_{G'}(C)$.*

Lemma 7.16 (Johnson's Trick, [Joh77]). *Let G be a directed graph, and let ϕ be an arbitrary potential function. Then $w_\phi(P) = w(P) + \phi(u) - \phi(v)$ for any u-v-path P, and $w_\phi(C) = w(C)$ for any cycle C. It follows that G and G_ϕ are equivalent.*

Lemma 7.17 ([Joh77]). *Let G be a directed graph without negative cycles and let $s \in V$ be a source vertex that can reach every other node. Then, for the potential ϕ defined as $\phi(v) = \mathrm{dist}_G(s,v)$, it holds that $w_\phi(e) \ge 0$ for all edges $e \in E$.*

Machine Model We work in the standard word RAM model with word size $\Theta(\log n + \log M)$, where n is the number of vertices and M is an upper bound on the largest edge weight in absolute value. That is, we assume that we can store vertex identifiers and edge weights in a single machine word, and perform basic operations in unit time.

8 Negative Weight SSSP

The goal of this chapter is to prove Main Theorem 7.1.

Organization The content is organized as follows. In Section 8.1 we present our algorithm for negative-weight SSSP on restricted graphs. In Section 8.2 we extend the algorithm from the previous section to work on general graphs without negative cycles. Finally, in Section 8.3 we remove this assumption and strengthen the algorithm to find negative cycles without worsening the running time.

Pedagogical note Most of the ideas at the core of our full algorithm can already be seen at play under the assumption that the graph does not have negative cycles. Moreover, under that assumption, the algorithm is simple, clean and hopefully easy to understand. Thus, we encourage the reader to focus on Section 8.1 and Section 8.2 to obtain the main ideas. Section 8.3, where we deal with negative cycles, is quite technical and most of the work goes into shaving one log factor (for pedagogical purposes, we give a simpler but slower version of an algorithm to find negative cycles in Section 8.3.2).

8.1 SSSP on Restricted Graphs

In this section we give an efficient algorithm for SSSP on restricted graphs (recall Definition 7.6). Specifically, we prove the following theorem:

Theorem 8.1 (Restricted SSSP)**.** *In a restricted graph G with source vertex $s \in V(G)$, we can compute a shortest path tree from s in time $O((m + n \log\log n)\log^2 n)$ with constant error probability $\frac{1}{2}$. (If the algorithm does not succeed, it returns* Fail*.)*

We develop this algorithm in two steps: First, we prove our decomposition scheme for restricted graphs (Section 8.1.1) and then we use the decomposition scheme to build an SSSP algorithm for restricted graphs (Section 8.1.2).

8.1.1 Decomposition for Restricted Graphs

In this section, we prove the decomposition lemma:

Lemma 7.8 (Decomposition)**.** *Let G be a restricted graph with source vertex $s \in V(G)$ and $\kappa \geq \kappa(G)$. There is a randomized algorithm* Decompose(G, κ) *running in expected time $O((m + n \log\log n)\log n)$ that computes an edge set $S \subseteq E(G)$ such that:*

1. Progress: *With high probability, for any strongly connected component C in $G \setminus S$, we have (i) $|C| \leq \frac{3}{4}|V(G)|$ or (ii) $\kappa(G[C \cup \{ s \}]) \leq \frac{\kappa}{2}$.*

2. Sparse Hitting: *For any shortest s-v-path P in G, we have $\mathbb{E}(|P \cap S|) \leq O(\log n)$.*

For the proof, we introduce some notation. Let $G_{\geq 0}$ denote the graph obtained by replacing negative edge weights by 0 in the graph G. A vertex v is *out-heavy* if $|B^{out}_{G_{\geq 0}}(v, \frac{\kappa}{4})| > \frac{n}{2}$ and *out-light* if $|B^{out}_{G_{\geq 0}}(v, \frac{\kappa}{4})| \leq \frac{3n}{4}$. Note that there can be vertices which are both out-heavy and out-light. We similarly define *in-light* and *in-heavy* vertices with "$B^{in}_{G_{\geq 0}}$" in place of "$B^{out}_{G_{\geq 0}}$".

Lemma 8.2 (Heavy-Light Classification)**.** *There is an algorithm which, given a directed graph G, labels every vertex correctly as either in-light or in-heavy (vertices which are both in-light and in-heavy may receive either label). The algorithm runs in time $O((m + n \log\log n) \log n)$ and succeeds with high probability.*

Note that by applying this lemma to the graph G^{rev} obtained by flipping the edge orientations, we can similarly classify vertices into out-light and out-heavy. We omit the proof for now as it follows easily from Lemma 9.6 which we state and prove in Section 9.2.

We are ready to state the decomposition algorithm: First, label each vertex as out-light or out-heavy and as in-light or in-heavy using the previous lemma. Then, as long as G contains a vertex v which is labeled out-light or in-light (say it is out-light), we will carve out a ball around v. To this end, we sample a radius r from the geometric distribution $\mathrm{Geom}(20 \log n/\kappa)$, we cut the edges $\partial B^{out}_{G_{\geq 0}}(v, r)$ (that is, the set of edges leaving $B^{out}_{G_{\geq 0}}(v, r)$) and we remove all vertices in $B^{out}_{G_{\geq 0}}(v, r)$ from the graph. We summarize the procedure in Algorithm 13. In what follows, we prove correctness of this algorithm.

Lemma 8.3 (Sparse Hitting of Algorithm 13)**.** *Let P be a shortest s-v-path in G and let S be the output of* Decompose(G, κ). *Then $\mathbb{E}(|P \cap S|) \leq O(\log n)$.*

Proof. Focus on any edge $e = (x, y) \in E(G)$. We analyze the probability that $e \in S$. We first analyze the probability of e being included into S in Line 7 (and the same analysis applies to the case where the edge is included in Line 11). Focus on any iteration of the loop in Line 5 for some out-light vertex v. There are three options:

- $x, y \notin B^{out}_{G_{\geq 0}}(v, r)$: The edge e is not touched in this iteration. It might or might not be included in later iterations.

- $x \in B^{out}_{G_{\geq 0}}(v, r)$ and $y \notin B^{out}_{G_{\geq 0}}(v, r)$: The edge e is contained in $\partial B^{out}_{G_{\geq 0}}(v, r)$ and thus definitely included into S.

- $y \in B^{out}_{G_{\geq 0}}(v, r)$: The edge e is definitely not included into S. Indeed, $e \notin \partial B^{out}_{G_{\geq 0}}(v, r)$, so we do not include e into S in this iteration. Moreover, as we remove y from G after this iteration, we will never consider the edge e again.

Algorithm 13 The graph decomposition. This algorithm $\textsc{Decompose}(G, \kappa)$ computes a subset of edges $S \subseteq E(G)$ satisfying the properties in Lemma 7.8.

```
procedure Decompose($G, \kappa$)
    Let $L^{in} \subseteq V(G)$ be the vertices labeled as in-light by Lemma 8.2 on $G$
    Let $L^{out} \subseteq V(G)$ be the vertices labeled as out-light by Lemma 8.2 on $G^{rev}$
    $S \leftarrow \emptyset$
    while there is $v \in V(G) \cap L^{out}$ do
        Sample $r \sim \mathrm{Geom}(20 \log n/\kappa)$
        $S \leftarrow S \cup \partial B^{out}_{G_{\geq 0}}(v, r)$
        $G \leftarrow G \setminus B^{out}_{G_{\geq 0}}(v, r)$
    while there is $v \in V(G) \cap L^{in}$ do
        Sample $r \sim \mathrm{Geom}(20 \log n/\kappa)$
        $S \leftarrow S \cup \partial B^{in}_{G_{\geq 0}}(v, r)$
        $G \leftarrow G \setminus B^{in}_{G_{\geq 0}}(v, r)$
    return $S$
```

Recall that the radius r is sampled from the geometric distribution $\mathrm{Geom}(p)$ for $p := 20 \log n/\kappa$. Therefore, we have that

$$\begin{aligned}
\mathbb{P}(e \in S) &\leq \max_{v \in V} \mathop{\mathbb{P}}_{r \sim \mathrm{Geom}(p)} (y \notin B^{out}_{G_{\geq 0}}(v, r) \mid x \in B^{out}_{G_{\geq 0}}(v, r)) \\
&= \max_{v \in V} \mathop{\mathbb{P}}_{r \sim \mathrm{Geom}(p)} (r < \mathrm{dist}_{G_{\geq 0}}(v, y) \mid r \geq \mathrm{dist}_{G_{\geq 0}}(v, x)) \\
&\leq \max_{v \in V} \mathop{\mathbb{P}}_{r \sim \mathrm{Geom}(p)} (r < \mathrm{dist}_{G_{\geq 0}}(v, x) + w_{G_{\geq 0}}(e) \mid r \geq \mathrm{dist}_{G_{\geq 0}}(v, x))
\end{aligned}$$

By the memoryless property of geometric distributions, we may replace r by the (non-negative) random variable $r' := r - \mathrm{dist}_{G_{\geq 0}}(v, x)$:

$$\begin{aligned}
&= \max_{v \in V} \mathop{\mathbb{P}}_{r' \sim \mathrm{Geom}(p)} (r' < w_{G_{\geq 0}}(e)) \\
&= \mathop{\mathbb{P}}_{r' \sim \mathrm{Geom}(p)} (r' < w_{G_{\geq 0}}(e)) \\
&\leq p \cdot w_{G_{\geq 0}}(e).
\end{aligned}$$

The last inequality follows since we can interpret $r' \sim \mathrm{Geom}(p)$ as the number of coin tosses until we obtain heads, where each toss is independent and lands heads with probability p. Thus, by a union bound, $\mathbb{P}(r' < w_{G_{\geq 0}}(e))$ is upper bounded by the probability that at least one of $w_{G_{\geq 0}}(e)$ coin tosses lands heads.

Now consider a shortest s-v-path P in G. Recall that $w_G(P) \leq 0$, since G is a restricted graph. Hence, P contains at most $\kappa(G) \leq \kappa$ edges with negative weight (i.e., with weight exactly -1). It follows that $w_{G_{\geq 0}}(P) \leq \kappa$ and thus finally:

$$\mathbb{E}(|P \cap S|) = \sum_{e \in P} \mathbb{P}(e \in S) = \sum_{e \in P} p \cdot w_{G_{\geq 0}}(e) \leq p \cdot w_{G_{\geq 0}}(P) \leq p\kappa = O(\log n). \qquad \square$$

In what follows, we will need the following lemma.

Lemma 8.4. *Let G be a directed graph. Then* $\min_C \bar{w}(C) = \min_Z \bar{w}(Z)$ *where C ranges over all cycles and Z ranges over all* closed walks *in G.*

Proof. Write $c = \min_C \bar{w}(C)$ and $z = \min_Z \bar{w}(Z)$. It suffices to prove that $c \leq z$. Take the closed walk Z witnessing z with the minimum number of edges. If Z is a cycle, then we clearly have $c \leq z$. Otherwise, Z must revisit at least one vertex and can therefore be split into two closed walks Z_1, Z_2. By the minimality of Z we have $\bar{w}(Z_1), \bar{w}(Z_2) > z$. But note that

$$z \cdot |Z| = w(Z) = w(Z_1) + w(Z_2) > z \cdot |Z_1| + z \cdot |Z_2| = z \cdot |Z|,$$

a contradiction. □

Lemma 8.5 (Progress of Algorithm 13)**.** *Let S be the output of* Decompose(G, κ). *Then, with high probability, any strongly connected component C in $G \setminus S$ satisfies (i) $|C| \leq \frac{3}{4}|V(G)|$ or (ii) $\kappa(G[C]) \leq \frac{\kappa}{2}$.*

Proof. Throughout, condition on the event that the heavy-light classification was successful (which happens with high probability). Observe that whenever we carve out a ball $B^{out}_{G_{\geq 0}}(v, r)$ and include its outgoing edges $\partial B^{out}_{G_{\geq 0}}(v, r)$ into S, then any two vertices $x \in B^{out}_{G_{\geq 0}}(v, r)$ and $y \notin B^{out}_{G_{\geq 0}}(v, r)$ cannot be part of the same strongly connected component in $G \setminus S$ (as there is no path from x to y). The same argument applies to $B^{in}_{G_{\geq 0}}(v, r)$.

Therefore, there are only two types of strongly connected components: (i) Those contained in $B^{out}_{G_{\geq 0}}(v, r)$ or $B^{in}_{G_{\geq 0}}(v, r)$, and (ii) those in the remaining graph after it no longer contains light vertices. We argue that each component of type (i) satisfies that $|C| \leq \frac{3}{4}|V(G)|$ (with high probability) and that each component of type (ii) satisfies $\kappa(G[C]) \leq \frac{\kappa}{2}$.

In case (i) we have $|C| \leq |B^{out}_{G_{\geq 0}}(v, r)|$. Since v is out-light, it follows that $|C| \leq \frac{3}{4}|V(G)|$ whenever $r \leq \frac{\kappa}{4}$. This event happens with high probability as:

$$\mathop{\mathbb{P}}_{r \sim \mathrm{Geom}(20 \log n/\kappa)} \left(r > \frac{\kappa}{4}\right) \leq \left(1 - \frac{20 \log n}{\kappa}\right)^{\frac{\kappa}{4}} \leq \exp(-5 \log n) \leq n^{-5}.$$

The number of iterations is bounded by n, thus by a union bound we never have $r > \frac{\kappa}{4}$ with probability at least $1 - n^{-4}$. A similar argument applies if we carve $B^{in}_{G_{\geq 0}}(v, r)$ when v is in-light.

Next, focus on case (ii). Let C be a strongly connected component in the remaining graph G after carving out all balls centered at light vertices. Suppose that $\kappa(G[C]) > \frac{\kappa}{2}$. We will construct a closed walk Z in G with mean weight $\bar{w}(Z) < 1$, contradicting the assumption that G is restricted by Lemma 8.4. Let P be the s-v-path in $G[C \cup \{ s \}]$ of nonpositive weight witnessing the largest number of negative edges (i.e., the path that witnesses $\kappa(G[C \cup \{ s \}])$), and let u be the first vertex (after s) on that path P. Let P_1 be the u-v-path obtained by removing the s-u-edge from P. Since the s-u-edge has weight 0, we have that $w(P_1) \leq 0$ and that P_1 contains more than $\frac{\kappa}{2}$ negative-weight edges. Since u, v are both out-heavy and in-heavy vertices in the original graph G, we have that $|B^{out}_{G_{\geq 0}}(v, \frac{\kappa}{4})|, |B^{in}_{G_{\geq 0}}(u, \frac{\kappa}{4})| > \frac{n}{2}$. It follows that these two balls must intersect and that there exists a v-u-path P_2 of length $w(P_2) \leq \frac{\kappa}{4} + \frac{\kappa}{4} = \frac{\kappa}{2}$. Combining P_1 and P_2, we obtain a closed walk Z with total weight $w(Z) \leq \frac{\kappa}{2}$ containing more than $\frac{\kappa}{2}$ (negative-weight) edges. It follows that $\bar{w}(Z) < 1$ yielding the claimed contradiction. ◻

Proof of Lemma 7.8. The correctness is immediate by the previous lemmas: Lemma 8.5 proves the progress property, and Lemma 8.3 the sparse hitting property. Next, we analyze the running time. Computing the heavy-light classification takes time $O((m + n \log\log n) \log n)$ due to Lemma 8.2. Sampling each radius r from the geometric distribution $\mathrm{Geom}(20 \log n/\kappa)$ runs in expected constant time in the word RAM with word size $\Omega(\log n)$ [BF13], so the overhead for sampling the radii is $O(n)$ in expectation. To compute the balls we use Dijkstra's algorithm. Using Thorup's priority queue [Tho04], each vertex explored in Dijkstra's takes time $O(\log\log n)$ and each edge time $O(1)$. Since every vertex contained in some ball is removed from subsequent iterations, a vertex participates in at most one ball. Note that a naive implementation of this would reinitialize the priority queue and distance array at each iteration of the while-loop. To avoid this, we initialize the priority queue and array of distances once, before the execution of the while-loops. Then, at the end of an iteration of the while-loop we reinitialize them in time proportional to the removed vertices and edges (this is the same approach as in the BNW algorithm [BNW22]). Thus, the overall time to compute all the balls is indeed $O(m + n \log\log n)$. ◻

8.1.2 Proof of Theorem 8.1

With the graph decomposition in hand, we can present our full algorithm for Restricted SSSP. The overall structure closely follows the BNW algorithm (see [BNW22, Algorithm 1]).

We start with the following crucial definition.

Definition 8.6. *Let G be a directed graph with a designated source vertex s. For any vertex $v \in V(G)$, we denote by $\eta_G(v)$ the smallest number of* negative-weight *edges in any shortest s-v-path.*

The next proposition captures the relationship between the parameters $\kappa(G)$ and $\eta_G(\cdot)$ when G is restricted (see Definitions 8.6 and 7.7).

Proposition 8.7. *Let G be a restricted graph with source vertex s. Then, for every vertex $v \in V$ it holds that $\eta_G(v) \leq \kappa(G)$.*

Proof. Fix a vertex v. Let P be a shortest s-v path witnessing $\eta_G(v)$ (see Definition 8.6). Since G is restricted, it does not contain negative cycles and thus P is a simple path. Furthermore, since there is an edge from s to v of weight 0, it follows that $w_G(P) \leq 0$.

Recall that $\kappa(G)$ is the maximum number of negative edges in any simple path which starts at s and has nonpositive weight (see Definition 7.7). Therefore, it follows that $\eta_G(v) \leq \kappa(G)$. □

Next, we use two lemmas from [BNW22]:

Lemma 8.8 (Dijkstra with Negative Weights, similar to [BNW22, Lemma 3.3])**.** *Let G be a directed graph with source vertex $s \in V(G)$ that does not contain a negative cycle. There is an algorithm that computes a shortest path tree from s in time $O(\sum_v (\deg(v) + \log\log n) \cdot \eta_G(v))$. (If G contains a negative cycle, the algorithm does not terminate.)*

The main differences to [BNW22, Lemma 3.3] are that we use a faster priority queue for Dijkstra and that [BNW22, Lemma 3.3] is restricted to graphs of constant maximum degree. Therefore, we devote Section 8.1.3 to a self-contained proof of Lemma 8.8.

Lemma 8.9 (DAG Edges, [BNW22, Lemma 3.2])**.** *Let G be a directed graph with nonnegative edge weights inside its SCCs. Then we can compute a potential function ϕ such that G_ϕ has nonnegative edge weights (everywhere) in time $O(n + m)$.*

Proof Sketch. For the complete proof, see [BNW22, Lemma 3.2]. The idea is to treat the graph as a DAG of SCCs, and to assign a potential function ϕ to every SCC such that the DAG edges become nonnegative. One way to achieve this is by computing a topological ordering, and by assigning $\phi(v)$ to be W times the rank of v's SCC in that ordering (here, $-W$ is the smallest weight in G). Then G_ϕ satisfies the claim. □

The Algorithm We are ready to state the algorithm; see Algorithm 14 for the pseudocode. Recall that $\kappa(G)$ is the maximum number of negative edges in any path P starting at s with $w(P) \leq 0$ (Definition 7.7). If $\kappa(G) \leq 2$, we run Lemma 8.8 to compute the distances from s. Otherwise, we start with applying our graph decomposition. That is, we compute a set of edges S, such that any strongly connected component C in the graph $G \setminus S$ is either small or has an improved κ-value. This constitutes enough progress to solve the induced graphs $G[C \cup \{ s \}]$ recursively. The recursive calls produce shortest path trees and thereby a potential function ϕ_1 such that G_{ϕ_1} has nonnegative edge weights inside each SCC. We then add back the missing edges by first calling Lemma 8.9 (to fix the edges $e \notin S$ between strongly connected components) and then Lemma 8.8 (to fix the edges $e \in S$). The correctness proof is easy:

Lemma 8.10 (Correctness of Algorithm 14)**.** *Let G be an arbitrary directed graph (not necessarily restricted), and let κ be arbitrary. Then, if* RestrictedSSSP(G, κ) *terminates, it correctly computes a shortest path tree from the designated source vertex s.*

Proof. If $\kappa \leq 2$ and the call in Line 3 terminates, then it correctly computes a shortest path tree due to Lemma 8.8. If $\kappa > 2$, then in Line 10 we compute a potential function ϕ_2 and in Line 11 we run Lemma 8.8 to compute a shortest path tree in the graph G_{ϕ_2}. Assuming that Lemma 8.8 terminates, this computation is correct since G_{ϕ_2} is equivalent to G. □

Algorithm 14 Solves the negative-weight SSSP problem on restricted graphs. The procedure RestrictedSSSP(G, κ) takes a restricted graph G and an upper bound $\kappa \geq \kappa(G)$, and computes a shortest path tree from the designated source vertex s.

```
1  procedure RestrictedSSSP($(G, \kappa)$)
2      if $\kappa \leq 2$ then
3          Run BF-Dijkstra on $G$ from $s$ and return the computed shortest path tree
4      Compute $S \leftarrow$ Decompose$(G, \kappa)$
5      Compute the strongly connected components $C_1, \dots, C_\ell$ of $G \setminus S$
6      for $i \leftarrow 1, \dots, \ell$ do
7          if $|C_i| \leq \frac{3n}{4}$ then $\kappa_i \leftarrow \kappa$ else $\kappa_i \leftarrow \frac{\kappa}{2}$
8          Recursively call RestrictedSSSP$(G[C_i \cup \{ s \}], \kappa_i)$
9          Let $\phi_1(v) = \mathrm{dist}_{G[C_i \cup \{ s \}]}(s, v)$ for all $v \in C_i$
10     Fix the DAG edges on $(G \setminus S)_{\phi_1}$ to obtain a potential $\phi_2$
11     Run BF-Dijkstra on $G_{\phi_2}$ and return the computed shortest path tree
```

Lemma 8.11 (Running Time of Algorithm 14)**.** *Let G be a restricted graph with $\kappa(G) \leq \kappa$. Then* RestrictedSSSP(G, κ) *runs in expected time $O((m + n \log\log n) \log^2 n)$.*

Proof. We first analyze the running time of a single call to Algorithm 14, ignoring the time spent in recursive calls. For the base case, when $\kappa(G) \leq 2$, the running time of Line 3 is $O(m + n \log\log n)$ by Lemma 8.8 and Proposition 8.7. Otherwise, the call to Decompose(G, κ) in Line 4 runs in time $O((m + n \log\log n) \log n)$ by Lemma 7.8. Computing the strongly connected components in $G \setminus S$ is in linear time $O(m + n)$, and so is the call to Lemma 8.9 in Line 10.

Analyzing the running time of Line 11 takes some more effort. Recall that $\eta_{G_{\phi_2}}(v)$ is the minimum number of negative edges in any s-v path in G_{ϕ_2} (see Definition 8.6). Our intermediate goal is to bound $\mathbb{E}(\eta_{G_{\phi_2}}(v)) = O(\log n)$ for all vertices v. Let S be the set of edges computed by the decomposition, as in the algorithm. We proceed in three steps:

- *Claim 1: $G_{\phi_1} \setminus S$ has nonnegative edges inside its SCCs.* The recursive calls in Line 8 correctly compute the distances by Lemma 8.10. Hence, for any two nodes $u, v \in C_i$, we have that $w_{\phi_1}(u, v) = w(u, v) + \mathrm{dist}_{G[C_i \cup \{ s \}]}(s, u) - \mathrm{dist}_{G[C_i \cup \{ s \}]}(s, v) \geq 0$, by the triangle inequality.

- *Claim 2: $G_{\phi_2} \setminus S$ has only nonnegative edges.* This is immediate by Lemma 8.9.

- *Claim 3: For every node v we have $\mathbb{E}(\eta_{G_{\phi_2}}(v)) \leq O(\log n)$.* Let P be a shortest s-v-path in G. Since G and G_{ϕ_2} are equivalent, P is also a shortest path in G_{ϕ_2}. By the previous claim, the only candidate negative edges in P are the edges in S. Therefore, we have that $\mathbb{E}(\eta_{G_{\phi_2}}(v)) \leq \mathbb{E}(|P \cap S|) = O(\log n)$, by Lemma 7.8.

The expected running time of Line 11 is thus bounded by

$$O\left(\sum_{v\in V(G)} (\deg(v) + \log\log n) \cdot \mathbb{E}(\eta_{G_{\phi_2}}(v))\right) = O\left(\sum_{v\in V(G)} (\deg(v) + \log\log n) \cdot \log n\right)$$
$$= O((m + n\log\log n)\log n).$$

Therefore, a single execution of Algorithm 14 runs in time $O((m + n\log\log n)\log n)$; let c denote the hidden constant in the O-notation.

We finally analyze the total running time, taking into account the recursive calls. We inductively prove that the running time is bounded by $c(m+n\log\log n)\log n\cdot\log_{4/3}(n\kappa)$.

We claim that for each recursive call on a subgraph $G[C_i\cup\{s\}]$, where C_i is a strongly connected component in $G\setminus S$, it holds that (i) $G[C_i\cup\{s\}]$ is a restricted graph and that (ii) $\kappa(G[C_i\cup\{s\}]) \le \kappa_i$. To see (i), observe that any subgraph of G containing s is also restricted. To show (ii), we distinguish two cases: Either $|C_i| \le \frac{3n}{4}$, in which case we trivially have $\kappa(G[C_i\cup\{s\}]) \le \kappa(G) \le \kappa = \kappa_i$. Or $|C_i| > \frac{3n}{4}$, and in this case Lemma 7.8 guarantees that $\kappa(G[C_i\cup\{s\}]) \le \frac{\kappa}{2} = \kappa_i$. It follows by induction that each recursive call runs in time $c\cdot(|E(G[C_i\cup\{s\}])| + |C_i|\log\log n)\log n\cdot\log_{4/3}(|C_i|\kappa_i)$. Moreover, observe that in either case we have $|C_i|\kappa_i \le \frac{3}{4}n\kappa$. Therefore, the total time can be bounded by

$$\begin{aligned}
&c(m + n\log\log n)\log n + \sum_{i=1}^{\ell} c\cdot(|E(G[C_i\cup\{s\}])| + |C_i|\log\log n)\log n\cdot\log_{4/3}(|C_i|\kappa_i)\\
&\quad\le c(m + n\log\log n)\log n\\
&\qquad + \sum_{i=1}^{\ell} c\cdot(|E(G[C_i\cup\{s\}])| + |C_i|\log\log n)\log n\cdot(\log_{4/3}(n\kappa) - 1)\\
&\quad\le c(m + n\log\log n)\log n + c(m + n\log\log n)\log n\cdot(\log_{4/3}(n\kappa) - 1)\\
&\quad= cm\log n\log\log n\cdot\log_{4/3}(n\kappa),
\end{aligned}$$

where in the third step we used that $\sum_i |E(G[C_i\cup\{s\}])| \le m$ and that $\sum_i |C_i| \le n$. This completes the running time analysis. □

Proof of Theorem 8.1. This proof is almost immediate from the previous two Lemmas 8.10 and 8.11. In combination, these lemmas prove that Algorithm 14 is a Las Vegas algorithm for the Restricted SSSP problem which runs in expected time $O((m + n\log\log n)\log^2 n)$. By interrupting the algorithm after twice its expected running time (and returning Fail in that case), we obtain a Monte Carlo algorithm with worst-case running time $O((m + n\log\log n)\log^2 n)$ and error probability $\frac{1}{2}$ as claimed. □

We remark that Algorithm 14 is correct even if the input graph G is not restricted—therefore, whenever G contains a negative cycle, the algorithm cannot terminate.

8.1.3 Lazy Dijkstra

This section is devoted to a proof of the following lemma, stating that Dijkstra's algorithm can be adapted to work with negative edges in time depending on the $\eta_G(v)$ values. Recall that $\eta_G(v)$ denotes the minimum number of negative-weight edges in a shortest s-v path in G.

Lemma 8.8 (Dijkstra with Negative Weights, similar to [BNW22, Lemma 3.3])**.** *Let G be a directed graph with source vertex $s \in V(G)$ that does not contain a negative cycle. There is an algorithm that computes a shortest path tree from s in time $O(\sum_v (\deg(v) + \log\log n) \cdot \eta_G(v))$. (If G contains a negative cycle, the algorithm does not terminate.)*

This lemma is basically [BNW22, Lemma 3.3], but the statement differs slightly. We provide a self-contained proof that morally follows the one in [BNW22, Appendix A].

We give the pseudocode for Lemma 8.8 in Algorithm 15. Throughout, let $G = (V, E, w)$ be the given directed weighted graph with possibly negative edge weights. We write $E^{\geq 0}$ for the subset of edges with nonnegative weight, and $E^{<0}$ for the subset of edges with negative weight. In the pseudocode, we rely on Thorup's priority queue:

Lemma 8.12 (Thorup's Priority Queue [Tho04])**.** *There is a priority queue implementation for storing n integer keys that supports the operations* FindMin*,* Insert *and* DecreaseKey *in constant time, and* Delete *in time $O(\log\log n)$.*

For the analysis of the algorithm, we define two central quantities. Let v be a vertex, then we define

$$\begin{aligned}\mathrm{dist}_i(v) &= \min\{\, w(P) : P \text{ is an } s\text{-}v\text{-path containing less than } i \text{ negative edges}\,\},\\ \mathrm{dist}'_i(v) &= \min\left\{\mathrm{dist}_i(v),\ \min_{\substack{u\in V\\ w(u,v)<0}} \mathrm{dist}_i(u) + w(u,v)\right\}.\end{aligned}$$

Note that $\mathrm{dist}_0(v) = \mathrm{dist}'_0(v) = \infty$. We start with some observations involving these quantities dist_i and dist'_i:

Observation 8.13. *For all i, $\mathrm{dist}_i(v) \geq \mathrm{dist}'_i(v) \geq \mathrm{dist}_{i+1}(v)$.*

Observation 8.14. *For all v,*

$$\mathrm{dist}_{i+1}(v) = \min\left\{\mathrm{dist}_i(v),\ \min_{\substack{u\in V\\ \mathrm{dist}_i(u)>\mathrm{dist}'_i(u)}} \mathrm{dist}'_i(u) + \mathrm{dist}_{G^{\geq 0}}(u,v)\right\}.$$

Proof. The statement is clear if $\mathrm{dist}_i(v) = \mathrm{dist}_{i+1}(v)$, so assume that $\mathrm{dist}_{i+1}(v) < \mathrm{dist}_i(v)$. Let P be the path witnessing $\mathrm{dist}_{i+1}(v)$, i.e., a shortest s-v-path containing less than $i+1$ negative edges. Let (x, u) denote the last negative-weight edge in P, and partition the path P into subpaths $P_1\, x\, u\, P_2$. Then the first segment $P_1\, x$ is a path containing less

Algorithm 15 The version of Dijkstra's algorithm implementing Lemma 8.8.

```
1  Initialize $d[s] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all vertices $v \neq s$
2  Initialize a Thorup priority queue $Q$ with keys $d[\cdot]$ and add $s$ to $Q$
3  repeat
       (The Dijkstra phase)
4      $A \leftarrow \emptyset$
5      while $Q$ is nonempty do
6          Remove the vertex $v$ from $Q$ with minimum $d[v]$
7          Add $v$ to $A$
8          for each edge $(v,x) \in E^{\geq 0}$ do
9              if $d[v] + w(v,w) < d[x]$ then
10                 Add $x$ to $Q$
11                 $d[x] \leftarrow d[v] + w(v,x)$
       (The Bellman-Ford phase)
12     for each $v \in A$ do
13         for each edge $(v,x) \in E^{<0}$ do
14             if $d[v] + w(v,x) < d[x]$ then
15                 Add $x$ to $Q$
16                 $d[x] \leftarrow d[v] + w(v,x)$
17 until $Q$ is empty
18 return $d[v]$ for all vertices $v$
```

than i negative-weight edges and the segment $u\,P_2$ does not contain any negative-weight edges. Therefore,

$$\mathrm{dist}_{i+1}(v) = \mathrm{dist}_i(x) + w(x,u) + \mathrm{dist}_{G^{\geq 0}}(u,v) \geq \mathrm{dist}'_i(u) + \mathrm{dist}_{G^{\geq 0}}(u,v).$$

Suppose, for the sake of contradiction, that $\mathrm{dist}_i(u) = \mathrm{dist}'_i(u)$. Then

$$\mathrm{dist}_{i+1}(v) \geq \mathrm{dist}_i(u) + \mathrm{dist}_{G^{\geq 0}}(u,v) \geq \mathrm{dist}_i(v),$$

which contradicts our initial assumption. □

Observation 8.15. *For all v,*

$$\mathrm{dist}'_i(v) = \min\left\{ \mathrm{dist}_i(v), \quad \min_{\substack{u \in V \\ \mathrm{dist}_{i-1}(u) > \mathrm{dist}_i(u) \\ w(u,v) < 0}} \mathrm{dist}_i(u) + w(u,v) \right\}$$

Proof. The statement is clear if $\mathrm{dist}_i(v) = \mathrm{dist}'_i(v)$, so suppose that $\mathrm{dist}'_i(v) < \mathrm{dist}_i(v)$. Then there is some vertex $u \in V$ with $w(u,v) < 0$ such that $\mathrm{dist}'_i(v) = \mathrm{dist}_i(u) + w(u,v)$.

It suffices to prove that $\mathrm{dist}_{i-1}(u) > \mathrm{dist}_i(u)$. Suppose for the sake of contradiction that $\mathrm{dist}_{i-1}(u) = \mathrm{dist}_i(u)$. Then $\mathrm{dist}'_i(v) = \mathrm{dist}_{i-1}(u) + w(u,v) \geq \mathrm{dist}'_{i-1}(v)$, which contradicts our initial assumption (by Observation 8.13). □

Lemma 8.16 (Invariants of Algorithm 15)**.** *Consider the i-th iteration of the loop in Algorithm 15 (starting at 1). Then the following invariants hold:*

1. *After the Dijkstra phase (after Line 11):*
   a. *$d[v] = \mathrm{dist}_i(v)$ for all vertices v, and*
   b. *$A = \{\, v : \mathrm{dist}_{i-1}(v) > \mathrm{dist}_i(v) \,\}$.*
2. *After the Bellman-Ford phase (after Line 16):*
   a) *$d[v] = \mathrm{dist}'_i(v)$ for all vertices v, and*
   b) *$Q = \{\, v : \mathrm{dist}_i(v) > \mathrm{dist}'_i(v) \,\}$.*

Proof. We prove the invariants by induction on i.

First Dijkstra Phase We start with the analysis of the first iteration, $i = 1$. The execution of the Dijkstra phase behaves exactly like the regular Dijkstra algorithm. It follows that $d[v] = \mathrm{dist}_{G^{\geq 0}}(s,v) = \mathrm{dist}_1(v)$, as claimed in Invariant 1a. Moreover, we include in A exactly all vertices which were reachable from s in $G^{\geq 0}$. Indeed, for these vertices v we have that $\mathrm{dist}_1(v) = \mathrm{dist}_{G^{\geq 0}}(s,v) < \infty$ and $\mathrm{dist}_0(v) = \infty$, and thus $A = \{\, v : \mathrm{dist}_0(v) > \mathrm{dist}_1(v) \,\}$, which proves Invariant 1b.

Later Dijkstra Phase Next, we analyze the Dijkstra phase for a later iteration, $i > 1$. Letting d' denote the state of the array d after the Dijkstra phase, our goal is to prove that $d'[v] = \mathrm{dist}_i(v)$ for all vertices v. So fix any vertex v; we may assume that $\mathrm{dist}_{i+1}(v) < \mathrm{dist}_i(v)$, as otherwise the statement is easy using that the algorithm never increases $d[\cdot]$. A standard analysis of Dijkstra's algorithm reveals that

$$d'[v] = \min_{u \in Q}(d[u] + \mathrm{dist}_{G^{\geq 0}}(u,v)),$$

where Q is the queue before the execution of Dijkstra. By plugging in the induction hypothesis and Observation 8.14, we obtain that indeed

$$d'[v] = \min_{\substack{u \in V \\ \mathrm{dist}_{i-1}(v) > \mathrm{dist}'_{i-1}(v)}} d[u] + \mathrm{dist}_{G^{\geq 0}}(u,v) = \mathrm{dist}_i(v),$$

which proves Invariant 1a.

To analyze Invariant 1b and the set A, first recall that we reset A to an empty set before executing the Dijkstra phase. Afterwards, we add to A exactly those vertices that are either (i) contained in the queue Q initially or (ii) for which $d'[v] < d[v]$. Note that these sets are exactly (i) $\{\, v : \mathrm{dist}_i(v) > \mathrm{dist}'_i(v) \,\}$ and (ii) $\{\, v : \mathrm{dist}'_{i-1}(v) > \mathrm{dist}_i(v) \,\}$ whose union is exactly $\{\, v : \mathrm{dist}_{i-1}(v) > \mathrm{dist}_i(v) \,\}$ by Observation 8.13.

Bellman-Ford Phase The analysis of the Bellman-Ford phase is simpler. Writing again d' for the state of the array d after the execution of the Bellman-Ford phase, by Observation 8.15 we have that

$$d'[v] = \min_{\substack{u \in A \\ w(u,v) < 0}} d[u] + w(u,v) = \min_{\substack{u \in V \\ \mathrm{dist}'_{i-1}(u) > \mathrm{dist}_i(u) \\ w(u,v) < 0}} \mathrm{dist}_i(u) + w(u,v) = \mathrm{dist}'_i(v),$$

which proves Invariant 2a. Here again we have assumed that $\mathrm{dist}'_i(v) < \mathrm{dist}_i(v)$, as otherwise the statement is trivial since the algorithm never increases $d[\cdot]$.

Moreover, after the Dijkstra phase has terminated, the queue Q was empty. Afterwards, in the current Bellman-Ford phase, we have inserted exactly those vertices v into the queue for which $\mathrm{dist}_i(v) > \mathrm{dist}'_i(v)$ and thus $Q = \{ v : \mathrm{dist}_i(v) > \mathrm{dist}'_i(v) \}$, which proves Invariant 2b. □

From these invariants (and the preceding observations), we can easily conclude the correctness of Algorithm 15:

Lemma 8.17 (Correctness of Algorithm 15)**.** *If the given graph G contains a negative cycle, then Algorithm 15 does not terminate. Moreover, if Algorithm 15 terminates, then it has correctly computed $d[v] = \mathrm{dist}_G(s,v)$.*

Proof. We show that after the algorithm has terminated, all edges (u,v) are *relaxed*, meaning that $d[v] \le d[u] + w(u,v)$. Indeed, suppose there is an edge (u,v) which is not relaxed, i.e., $d[v] > d[u] + w(u,v)$. Let i denote the final iteration of the algorithm. By Invariant 2a we have that $d[x] = \mathrm{dist}'_i(x)$ and by Invariant 2b we have that $\mathrm{dist}'_i(x) = \mathrm{dist}_i(x)$ (assuming that $Q = \emptyset$), for all vertices x. We distinguish two cases: If $w(u,v) \ge 0$, then we have that $\mathrm{dist}_i(v) > \mathrm{dist}_i(u) + w(u,v)$—a contradiction. And if $w(u,v) < 0$, then we have that $\mathrm{dist}'_i(v) = \mathrm{dist}_i(u) + w(u,v)$—also a contradiction.

So far we have proved that if the algorithm terminates, all edges are relaxed. It is easy to check that if G contains a negative cycle, then at least one edge in that cycle cannot be relaxed. It follows that the algorithm does not terminate whenever G contains a negative cycle.

Instead, assume that G does not contain a negative cycle. We claim that the algorithm has correctly computed all distances. First, recall that throughout we have $d[v] \ge \mathrm{dist}_G(s,v)$. Consider any shortest s-v-path P; we prove that $d[v] = w(P)$ by induction on the length of P. For $|P| = 0$, we have correctly set $d[s] = 0$ initially. (Note that $\mathrm{dist}_G(s,s)$ cannot be negative as otherwise G would contain a negative cycle.) So assume that P is nonempty and that P can be written as $P_1\, u\, v$. Then by induction $d[u] = \mathrm{dist}_G(P_1\, u)$. Since the edge (u,v) is relaxed, we have that $d[v] \le d[u] + w(u,v) = w(P) = \mathrm{dist}_G(s,v)$. Recall that we also have $d[v] \ge \mathrm{dist}_G(s,v)$ and therefore $d[v] = \mathrm{dist}_G(s,v)$. □

For us, the most relevant change in the proof is the running time analysis. Recall that $\eta_G(v)$ denotes the minimum number of negative edges in a shortest s-v-path, and that $\deg(v)$ denotes the out-degree of a vertex v.

Lemma 8.18 (Running Time of Algorithm 15)**.** *Assume that G does not contain a negative cycle. Then Algorithm 15 runs in time $O(\sum_v (\deg(v) + \log\log n)\eta_G(v))$.*

Proof. Consider a single iteration of the algorithm. Letting A denote the state of the set A at the end of (Dijkstra's phase of) the iteration, the running time of the whole iteration can be bounded by:

$$O\left(\sum_{v \in A} (\deg(v) + \log\log n)\right).$$

Indeed, in the Dijkstra phase, in each iteration we spend time $O(\log\log n)$ for deleting an element from the queue (Lemma 8.12), but for each such deletion in Q we add a new element to A. Moreover, both in the Dijkstra phase and the Bellman-Ford phase we only enumerate edges starting from a vertex in A, amounting for a total number of $O(\sum_{v\in A} \deg(v))$ edges. The inner steps of the loops (in Lines 9 to 11 and Lines 14 to 16) run in constant time each (Lemma 8.12).

Let us write A_i for the state of i in the i-th iteration. Then the total running time is

$$O\left(\sum_{i=1}^{\infty} \sum_{v \in A_i} (\deg(v) + \log\log n)\right) = O\left(\sum_{v \in V} |\{\, i : v \in A_i \,\}| \cdot (\deg(v) + \log\log n)\right).$$

To complete the proof, it suffices to show that $|\{\, i : v \in A_i \,\}| \leq \eta_G(v)$. To see this, we first observe that $\mathrm{dist}_{\eta_G(v)+1}(v) = \mathrm{dist}_{\eta_G(v)+2} = \cdots = \mathrm{dist}_G(s, v)$. Since, by the invariants above we know that $A_i = \{\, v : \mathrm{dist}_{i-1}(v) > \mathrm{dist}_i(v) \,\}$, it follows that v can only be contained in the sets $A_1, \ldots, A_{\eta_G(v)}$. □

In combination, Lemmas 8.17 and 8.18 complete the proof of Lemma 8.8.

8.2 SSSP on Graphs without Negative Cycles

In this section we present the $O((m + n\log\log n)\log^2(n)\log(nW))$-time algorithm for SSSP on graphs G without negative cycles. Later in Section 8.3, we will remove the assumption that G does not contain negative cycles, and strengthen the algorithm to find a negative cycle if it exists.

The main idea is to use *scaling* and some tricks for probability amplification in order to extend our algorithm for restricted graphs developed in Section 8.1. More precisely, we use the standard *scaling technique* [Gab83; GT89; Gol95; BNW22] to reduce the computation of SSSP in an arbitrary graph (without negative cycles) to the case of restricted graphs. Formally, we prove the following theorem:

Theorem 8.19 (Scaling Algorithm for SSSP)**.** *There is a Las Vegas algorithm which, given a directed graph G without negative cycles and with a source vertex $s \in V(G)$, computes a shortest path tree from s, running in time $O(T_{\mathrm{RSSSP}}(m, n) \cdot \log(nW))$ with high probability (and in expectation).*

Algorithm 16 One step of the scaling algorithm. Given a graph G with minimum weight greater than $-3W$, $\textsc{Scale}(G)$ computes a potential ϕ such that G_ϕ has minimum weight greater than $-2W$. See Lemma 8.20.

```
1  procedure $\textsc{Scale}(G)$
2      Let $W$ be such that all weights in $G$ are greater than $-3W$
3      Let $H$ be a copy of $G$ with edge weights $w_H(e) = \lceil w_G(e)/W \rceil + 1$, and add
           an artificial source vertex $s$ to $H$ with weight-0 edges to all other vertices
4      Compute a shortest path tree from $s$ in the restricted graph $H$ using Theorem 8.1
5      Let $\phi$ be the potential defined by $\phi(v) = W \cdot \mathrm{dist}_H(s, v)$
6      return $\phi$
```

One-Step Scaling The idea of the scaling algorithm is to increase the smallest weight in G step-by-step, while maintaining an equivalent graph. The following Lemma 8.20 gives the implementation of one such scaling step as a direct reduction to Restricted SSSP.

Lemma 8.20 (One-Step Scaling)**.** *Let G be a directed graph that does not contain a negative cycle and with minimum weight greater than $-3W$ (for some integer $W \geq 1$). There is an algorithm $\textsc{Scale}(G)$ computing ϕ such that G_ϕ has minimum weight greater than $-2W$, which succeeds with constant probability (if the algorithm does not succeed, it returns $\textsc{Fail}$) and runs in time $O(T_{RSSSP}(m, n))$.*

Proof. We construct a restricted graph H as a copy of G with modified edge weights $w_H(e) = \lceil w_G(e)/W \rceil + 1$. We also add a source vertex s to H, and put edges of weight 0 from s to all other vertices. We compute a shortest path tree from s in H using Theorem 8.1, and return the potential ϕ defined by $\phi(v) = W \cdot \mathrm{dist}_H(s, v)$. For the pseudocode, see Algorithm 16. Note that the running time is dominated by computing shortest paths in a restricted graph.

To prove that the algorithm is correct, we first check that H is indeed restricted (see Definition 7.6):

- Each edge weight satisfies $w_H(e) = \lceil w_G(e)/W \rceil + 1 \geq \lceil (-3W + 1)/W \rceil + 1 = -1$.
- Consider any cycle C in H. Recall that $w_G(C) \geq 0$ (as G does not contain negative cycles), and thus

$$\bar{w}_H(C) = \frac{w_H(C)}{|C|} = \frac{1}{|C|} \sum_{e \in C} w_H(e) = 1 + \frac{1}{|C|} \sum_{e \in C} \left\lceil w_G(e) \cdot \frac{1}{W} \right\rceil \geq 1 + \frac{w_G(C)}{W|C|} \geq 1.$$

  In particular, the minimum cycle mean in H is at least 1.
- Finally, we have artificially added a source vertex s to H with weight-0 edges to all other vertices.

Algorithm 17 The fast SSSP algorithm. In a given graph G without negative cycles, it computes a shortest path tree in G from the given source vertex s.

```
1   procedure SSSP($G, s$)
2       Let $-W$ be the smallest edge weight in $G$
3       Let $G_0$ be a copy of $G$ with edge weights $w_{G_0}(e) = 4n \cdot w_G(e)$
4       for $i \leftarrow 0, \dots, L-1$ where $L = \Theta(\log(nW))$ do
5           $\phi_i \leftarrow$ Scale($G_i$) (rerun the algorithm until it succeeds)
6           $G_{i+1} \leftarrow (G_i)_{\phi_i}$
7       Let $G^*$ be a copy of $G_L$ with negative weights replaced by 0
8       return a shortest path tree in $G^*$ from $s$, computed by Dijkstra's algorithm
```

It remains to prove that the potential ϕ defined by $\phi(v) = W \cdot \mathrm{dist}_H(s,v)$ satisfies that G_ϕ has minimum edge weight more than $-2W$. Consider any edge $e = (u,v)$. Since by definition $w_H(e) < w_G(e) \cdot \frac{1}{W} + 2$, we have that $w_G(e) > W \cdot (w_H(e) - 2)$. It follows that

$$\begin{aligned} w_{G_\phi}(e) &= w_G(e) + \phi(u) - \phi(v) \\ &= w_G(e) + W \cdot \mathrm{dist}_H(s,u) - W \cdot \mathrm{dist}_H(s,v) \\ &> -2W + W \cdot w_H(e) + W \cdot \mathrm{dist}_H(s,u) - W \cdot \mathrm{dist}_H(s,v) \\ &\geq -2W. \end{aligned}$$

In the last step we have used the triangle inequality $\mathrm{dist}_H(s,v) \leq \mathrm{dist}_H(s,u) + w_H(u,v)$.

Finally, we argue that the algorithm succeeds with constant probability. Observe that the algorithm succeeds if the computation of the shortest path tree from s succeeds in Line 4 (indeed, all other steps are deterministic). Since H is restricted, Theorem 8.1 guarantees that this holds with constant probability, and if it does not succeed it returns Fail, completing the proof. □

The Complete Scaling Algorithm We are ready to state the algorithm SSSP(G, s) which implements Theorem 8.19. We construct a graph G_0 by multiplying every edge weight of G by $4n$. Then, for $i = 0, \dots, L-1$ where $L = \Theta(\log(nW))$, we call Scale(G_i) (we repeat the call until it succeeds) to obtain a potential ϕ_i and set $G_{i+1} := (G_i)_{\phi_i}$. Next, we construct a graph G^* as a copy of G_L, with every negative edge weight replaced by 0. Finally, we compute a shortest path tree in G^* using Dijkstra's algorithm. For the details, see the pseudocode in Algorithm 17.

Lemma 8.21 (Running Time of Algorithm 17)**.** *If G does not contain a negative cycle, then SSSP(G, s) runs in time $O(T_{\mathrm{RSSSP}}(m,n) \cdot \log(nW))$ with high probability (and in expectation).*

Proof. We analyze the running time of the for-loop in Line 4, which runs for $L = O(\log(nW))$ iterations. Each iteration repeatedly calls Scale(G_i) until one such call succeeds. By Lemma 8.20, a single call succeeds with constant probability (say, $\frac{1}{2}$) and runs in time $O(T_{\text{RSSSP}}(m,n))$. We can therefore model the running time of the i-th iteration by $O(X_i \cdot T_{\text{RSSSP}}(m,n))$ where $X_i \sim \text{Geom}(\frac{1}{2})$ is a geometric random variable. Therefore, by Chernoff's bound, the time of the for-loop is bounded by $O(\sum_{i=0}^{L-1} X_i \cdot T_{\text{RSSSP}}(m,n)) = O(T_{\text{RSSSP}}(m,n) \cdot L)$ with probability at least $1 - \exp(-\Omega(L)) \geq 1 - n^{-\Omega(1)}$. Finally, observe that $T_{\text{RSSSP}}(m,n) = \Omega(m+n)$, and therefore the call to Dijkstra's algorithm in Line 8 is dominated by the time spent in the for-loop. □

Lemma 8.22 (Correctness of Algorithm 17)**.** *If G does not contain a negative cycle, then Algorithm 17 correctly computes a shortest path tree from s.*

Proof. Consider an execution of Algorithm 17. We prove that any shortest path in G^* is a shortest path in G, and hence the shortest path tree from s computed in G^* is also a shortest path tree from s in G, implying correctness. We proceed in three steps:

- As G_0 is a copy of G with scaled edge weights $w_{G_0}(e) = 4n \cdot w_G(e)$, any path P also has scaled weight $w_{G_0}(P) = 4n \cdot w_G(P)$ and therefore G and G_0 are equivalent.
- Since the graphs $G_0, \dots, G_L$ are obtained from each other by adding potential functions, they are equivalent (see Lemma 7.16). Moreover, by the properties of Lemma 8.20, the smallest weight $-W$ increases by a factor $\frac{2}{3}$ in every step until G_L has smallest weight at least -3. Here we use that $L = \Omega(\log(nW))$ for sufficiently large hidden constant.
- G^* is the graph obtained from G_L by replacing negative-weight edges by 0-weight edges. Consider any non-shortest u-v-path P' in G_L. We will show that P' is also not a shortest u-v path in G^*, which completes the argument. Towards that end, let P be any shortest u-v-path. Recall that G_L equals $(G_0)_\phi$ for some potential function ϕ. Therefore:

$$\begin{aligned} w_{G_L}(P') - w_{G_L}(P) &= w_{G_0}(P') + \phi(u) - \phi(v) - w_{G_0}(P) - \phi(u) + \phi(v) \\ &= w_{G_0}(P') - w_{G_0}(P) \\ &\geq 4n, \end{aligned}$$

  where the last inequality uses that the weights of P and P' in G_0 differ by at least $4n$ (this is why we scaled the edge weights by $4n$ in G_0). Finally, recall that by transitioning to G^* we can increase the weight of any path by at most $3 \cdot (n-1)$. It follows that

$$w_{G^*}(P') - w_{G^*}(P) \geq w_{G_L}(P') - w_{G_L}(P) - 3 \cdot (n-1) \geq 4n - 3 \cdot (n-1) > 0,$$

  and therefore, P' is not a shortest u-v-path in G^*. Hence, a shortest path in G^* is also a shortest path in G_L, and since G_L is equivalent to G, it is also a shortest path in G. □

The proof of Theorem 8.19 is immediate by combining Lemmas 8.21 and 8.22.

We end this section with the following lemma, which will be useful in the next section.

Lemma 8.23. *Let G be a directed weighted graph and $s \in V(G)$. If SSSP(G, s) terminates, then G does not contain negative cycles.*

Proof. Assume for the sake of contradiction that G has a negative cycle C and that SSSP(G, s) terminates. Consider the graph G_L which is constructed in the last iteration of the for-loop in Line 4. Note that G_L is equivalent to G_0, since it was obtained by adding potential functions. Observe that the weight of C in G_0 and G_L is at most $-4n$, since it was negative in G and we scaled by a factor $4n$ (see Lemma 7.16). Recall that we chose $L = \Theta(\log(nW))$ with large enough hidden constant so that the smallest weight in G_L is at least -3. This implies that the weight of the minimum cycle in G_L is at least $-3n$, a contradiction. □

8.3 Finding Negative Cycles

In Section 8.2 we developed an algorithm to compute a shortest path tree with high probability in a graph without negative cycles. In this section, we extend that result to *find* a negative cycle if it exists. As a warm-up, we observe that the SSSP algorithm developed in Theorem 8.19 can be used to *detect* the presence of a negative cycle with high probability:

Corollary 8.24. *Let G be a directed graph. There is an algorithm* DetectNegCycle(G) *with the following properties:*

- *If G has a negative cycle, then the algorithm reports* NegCycle.
- *If G does not have a negative cycle, then with high probability it returns* NoNegCycle
- *It runs in time $O(T_{RSSSP}(m, n) \log(nW))$.*

Proof. The algorithm adds a dummy source s connected with 0-weight edges to all vertices in G and runs SSSP(G, s). If it finishes within its time budget, we return NoNegCycle, otherwise we interrupt the computation and return NegCycle. The running time follows immediately by the guarantee of Theorem 8.19.

Now we argue about correctness. If G contains no negative cycles, then the algorithm returns NoNegCycle with high probability due to Theorem 8.19. If G contains a negative cycle, then Lemma 8.23 implies that SSSP(G, s) does not terminate, so in this case we always report NegCycle. □

Finding the negative cycle though, requires some more work. Towards this end, we follow the ideas of [BNW22]. They reduced the problem of finding a negative cycle to a problem called Threshold, which we define next. We will use the following notation: given a directed graph G and an integer M, we write G^{+M} to denote the graph obtained by adding M to every edge weight of G.

Definition 8.25 (Threshold)**.** *Given a directed graph G,* Threshold(G) *is the smallest integer $M^* \geq 0$ such that G^{+M^*} contains no negative cycle.*

Algorithm 18 The procedure to find a negative cycle. Given a graph G containing a negative cycle, it finds one such negative cycle C. See Lemma 8.26.

```
procedure FindNegCycle($G$)
    Let $G_0$ be a copy of $G$ with $w_{G_0}(e) = (n^3 + 1) \cdot w_G(e)$
    Let $M^* \leftarrow$ Threshold($G_0$)
    Let $G_1$ be a copy of $G_0^{+M^*}$ where we add an artificial source
        vertex $s$ to $G_1$ with weight-0 edges to all other vertices
    Run SSSP($G_1, s$) (see Algorithm 17) and set $\phi(v) = \mathrm{dist}_{G_1}(s, v)$
    Let $G_2$ be the graph $(G_1)_\phi$ where we remove all edges with weight greater than
  $n$
    if there is a cycle $C$ in $G_2$ then
        if $C$ is negative in $G$ then return $C$
    return FindNegCycle($G$) (i.e., restart)
```

For a graph G, we write $T_{\text{Threshold}}(m, n)$ for the optimal running time of an algorithm computing Threshold(G) with high probability.

The remainder of the section is organized as follows: in Section 8.3.1 we give the reduction from finding negative cycles to Threshold. In Section 8.3.2 we give an implementation of Threshold which has an extra log-factor compared to the promised Main Theorem 7.1, but it has the benefit of being simple. Finally, in Section 8.3.3 we give a faster (but more involved) implementation of Threshold which yields Main Theorem 7.1.

8.3.1 Reduction to Threshold

In this section we restate the reduction given by Bernstein et al. in [BNW22, Section 7.1] from finding a negative cycle if it exists, to Threshold and RestrictedSSSP (see their algorithm SPLasVegas).

Lemma 8.26 (Finding Negative Cycles). *Let G be a directed graph with a negative cycle. There is a Las Vegas algorithm FindNegCycle(G) which finds a negative cycle in G, and runs in time $O(T_{RSSSP}(m, n) \log(nW) + T_{\text{Threshold}}(m, n))$ with high probability.*

Proof. See the pseudocode in Algorithm 18 for a concise description. We start by defining a graph G_0 which is a copy of G but with edge weights multiplied by $n^3 + 1$. Then we compute M^* using Threshold(G_0), and let G_1 be $G_0^{+M^*}$. Next, we add a dummy source s to G_1 connected with 0-weight edges to all other vertices, and run SSSP on the resulting graph from s. We then use the distances computed to construct a potential ϕ, and construct a graph G_2 by applying the potential ϕ to G_1 and subsequently removing all the edges with weight larger than n. Finally, we check if G_2 contains any cycle (of any weight) and if so, check it has negative weight in the original graph G and return it. Otherwise, we restart the algorithm from the beginning.

The correctness is obvious: When the algorithm terminates, it clearly returns a negative cycle. The interesting part is to show that with high probability the algorithm finds a negative cycle C without restarting. The call to $\textsc{Threshold}(G_0)$ in Line 3 returns the smallest $M^* \geq 0$ such that G_0 contains no negative cycle, with high probability. In this case, by definition, G_1 does not contain a negative cycle, and therefore by Theorem 8.19 the call to $\text{SSSP}(G_1, s)$ correctly computes a shortest path tree from s. From now on, we condition on these two events.

▷ Claim 8.27. It holds that $M^* > n^2$.

Proof. Let C be a simple cycle in G with minimum (negative) weight. Since $G_1 = G_0^{+M^*}$ contains no negative cycles, the weight of C in G_1 is $0 \leq w_1(C) = w_0(C) + M^*|C|$. The claim follows by noting that $w_0(C) < -n^3$ due to the scaling in Line 2, and that $|C| \leq n$ because C is simple. ◁

Next, we argue that a cycle of minimum weight in G remains a cycle in G_2, and conversely that any simple cycle in G_2 corresponds to a negative weight cycle in G. Note that this is enough to prove that the algorithm terminates with high probability without a restart.

▷ Claim 8.28. Let C be a simple cycle in G of minimum weight. Then, C is a cycle in G_2.

Proof. First note that the weight of C in $G_0^{+M^*}$ (and thus also in G_1) is at most n. This holds since M^* is the smallest integer such that $G_0^{+M^*}$ contains no negative cycles, which means that $w_0(C) - |C| < 0$. Second, note that since Line 5 correctly computes a shortest path tree in G_1, it holds that the edge weights in $(G_1)_\phi$ are all nonnegative (by Lemma 7.17). Moreover, the weight of C in $(G_1)_\phi$ is the same as in G_1 (by Lemma 7.16). Thus, we conclude that the removal of the edges of weight greater than n in $(G_1)_\phi$ to obtain G_2 leaves C untouched. ◁

▷ Claim 8.29. Any cycle C' in G_2 has negative weight in G.

Proof. Note that $w_2(C') \leq n^2$ since every edge in G_2 has weight at most n. Moreover, since G_2 is obtained from G_1 by adding a potential, it holds that $w_2(C') = w_1(C')$ (by Lemma 7.16). Therefore, $w_0(C') = w_1(C') - M^*|C'| \leq n^2 - M^* < 0$ where the last inequality holds since $M^* > n^2$ by Claim 8.27. ◁

Finally, we analyze the running time. The call to $\textsc{Threshold}(G_0)$ succeeds with high probability (see Definition 8.25). Conditioned on this, G_1 contains no negative cycles. Thus by Theorem 8.19, the call to $\text{SSSP}(G, s)$ runs in time $O(T_{\text{RSSSP}}(m, n) \log(nW))$ with high probability. Note that the remaining steps of the algorithm take time $O(m)$. Therefore, we conclude that the overall running time is

$$O(T_{\text{RSSSP}}(m, n) \log(nW) + T_{\textsc{Threshold}}(m, n))$$

with high probability. □

8.3.2 Simple Implementation of Threshold

In this section we give a simple implementation of Threshold which combined with Lemma 8.26 yields an algorithm to find negative cycles in time

$$O(T_{\text{RSSSP}}(m,n)\log n\log(nW)).$$

This procedure shaves one log-factor compared to the analog in the BNW algorithm (see their procedure FindThresh in [BNW22, Lemma 7.2]). Later, in Section 8.3.3, we give an improved but more intricate algorithm.

As a building block, we will use the routine Scale from Lemma 8.20. The following lemma boosts the probability of success of Scale and uses a different parameterization of the minimum weight in the input graph, which will streamline our presentation.

Lemma 8.30 (Test Scale)**.** *Let G be a directed graph with minimum weight at least $-W$ where $W \geq 24$, and let $0 < \delta < 1$ be a parameter. There is an algorithm TestScale(G, δ) with the following properties:*

- *If G does not contain a negative cycle, then with probability at least $1-\delta$ it succeeds and returns a potential ϕ such that G_ϕ has minimum weight at least $-\frac{3}{4}W$. If it does not suceed, it returns Fail.*
- *It runs in time $O(T_{RSSSP}(m,n)\cdot\log(1/\delta))$.*

Proof. We run Scale(G) (see Lemma 8.20) for $O(\log(1/\delta))$ repetitions. Each execution either returns a potential ϕ, or it fails. We return Fail if and only if *all* these repetitions fail. The running time analysis is immediate by Lemma 8.20.

Now we analyze correctness. First we look at the success probability. Lemma 8.20 guarantees that if G does not contain a negative cycle, then each invocation to Scale(G) returns a potential ϕ with constant probability. Thus, in this case, the probability that all $O(\log(1/\delta))$ repetitions fail and we return Fail is at most δ, as stated. Next, we analyze the increase in the minimum weight of G_ϕ. Recall that the minimum weight in G is at least $-W$. Let k be the largest integer such that $W \geq 3k$, and let $-W'$ denote the minimum weight of G_ϕ. In particular, the minimum weight in G is greater than $-3(k+1)$, so Lemma 8.20 guarantees that

$$-W' > -2(k+1) \geq -\tfrac{2}{3}W - 2 \geq -\tfrac{2}{3}W - \tfrac{1}{12}W = -\tfrac{3}{4}W,$$

where the last inequality uses the assumption that $W \geq 24$. □

Lemma 8.31 (Slow Threshold)**.** *Let G be a directed graph. There is an algorithm computing Threshold(G) (Definition 8.25) which succeeds with high probability and runs in worst-case time $O(T_{RSSSP}(m,n)\log n\log(nW))$.*

Algorithm 19 The slow implementation of Threshold. Given a graph G, it computes the smallest integer $M^* \geq 0$ such that G^{+M^*} contains no negative cycle. See Lemma 8.31.

```
procedure SlowThreshold($G$)
    Let $-W$ be the smallest weight in $G$
    if $W \leq 48$ then
        for $t \leftarrow 47, \dots, 1, 0$ do
            if DetectNegCycle($G^{+t}$) = NegCycle then return $t + 1$
        return 0
    else
        Let $M \leftarrow \lceil \frac{W}{2} \rceil$
        if TestScale($G^{+M}, n^{-10}$) = $\phi$ then return SlowThreshold($G_\phi$)
        else return $M$ + SlowThreshold($G^{+M}$)
```

Proof. We summarize the pseudocode in Algorithm 19. Let $-W$ be the smallest weight in G. If $W \leq 48$ (i.e., all weights are at least -48) we clearly have that the correct answer lies in the range $0 \leq M^* \leq 48$. We brute-force the answer by exhaustively checking which graph $G^{+47}, \dots, G^{+0}$ is the first one containing a negative cycle. For this test we use the algorithm DetectNegCycle(G). Corollary 8.24 guarantees that it reports correct answers with high probability.

If $W > 48$, we make progress by reducing the problem to another instance with larger minimum weight. Let $M = \lceil \frac{W}{2} \rceil$, and run TestScale($G^{+M}, \delta$) for $\delta := 1/n^{10}$. We distinguish two cases based on the outcome of TestScale:

- Case 1: TestScale(G^{+M}, δ) = ϕ for a potential function ϕ. Then recursively compute and return SlowThreshold(G_ϕ). First note that this is correct, i.e., that the answer is unchanged by recursing on G_ϕ, since the potential does not change the weight of any cycle (see Lemma 7.16). Second, note that we make progress by increasing the smallest weight in G_ϕ to least $-\frac{11}{12}W$: To see this, note that the minimum weight of G^{+M} is at least $-\frac{1}{2}W$, and thus, Lemma 8.30 guarantees that the smallest weight in G_ϕ^{+M} is at least $-\frac{3}{8}W$. Therefore, it follows that the smallest weight in G_ϕ is at least

$$-\tfrac{3}{8}W - M = \tfrac{3}{8}W - \lceil \tfrac{1}{2}W \rceil \geq -\tfrac{7}{8}W - 1 > -\tfrac{7}{8}W - \tfrac{1}{24}W = -\tfrac{11}{12}W,$$

  where the second inequality uses the assumption that $W > 24$.

- Case 2: TestScale(G^{+M}, δ) = Fail. By Lemma 8.30, if G^{+M} does not contain a negative cycle then with high probability the output is not Fail. Conditioned on this event, we conclude that G^{+M} contains a negative cycle. Thus, we know that the optimal answer M^* satisfies $M^* \geq M$, and therefore we return M + SlowThreshold(G^{+M}). Note that this also improves the most negative edge weight to $-W + M \geq -\frac{11}{12}W$.

We claim that the running time is bounded by $O(T_{\text{RSSSP}}(m, n) \log n \log(nW))$. To see this, note that in the base case, when $W \leq 48$, the algorithm calls DetectNegCycle(G)

and therefore takes time $O(T_{\mathrm{RSSSP}}(m, n) \cdot \log(nW))$ (see Corollary 8.24). We claim that the higher levels of the recursion take time $O(T_{\mathrm{RSSSP}}(m, n) \log n \log W)$ in total. Note that each such level takes time $O(T_{\mathrm{RSSSP}}(m, n) \cdot \log n)$ due to the call to TestScale (Lemma 8.30) and thus, it suffices to bound the recursion depth by $O(\log W)$. To this end, observe that we always recur on graphs for which W has decreased by a constant factor.

Finally note that each call to TestScale succeeds with high probability, and we make one call for each of the $O(\log W)$ recursive calls. Thus, by a union bound the algorithm succeeds with high probability. (Strictly speaking, for this union bound we assume that $\log W \leq n$; if instead $\log W > n$, we can simply use Bellman-Ford's algorithm.) □

8.3.3 Fast Implementation of Threshold

In this section we give the fast implementation of Threshold.

Lemma 8.32 (Fast Threshold)**.** *Let G be a directed graph. There is an algorithm computing* Threshold(G) *(see Definition 8.25) which suceeds with high probability, and runs in worst-case time $O(T_{\mathrm{RSSSP}}(m, n) \log(nW))$.*

The algorithm is intricate, so we start with a high level description to convey some intuition.

High-Level Idea Let Δ be a parameter and let $M^* \geq 0$ be the right threshold. Let us look at what happens if we make a call to TestScale$(G^{+W-\Delta}, \delta)$, where $1-\delta$ is the success probability and $-W$ is the minimum edge weight in G. If $G^{+W-\Delta}$ does not have negative cycles, then Lemma 8.30 guarantees that with probability at least $1-\delta$ we obtain a potential ϕ. On the other hand, if $G^{+W-\Delta}$ contains a negative cycle, then we have *no guarantee* from Lemma 8.30. That is, the algorithm might return a potential, or it might return Fail. The upside is that as long as we obtain a potential, regardless whether there is a negative cycle or not, we can make progress by (additively) increasing the minimum edge weight by $\approx \Delta$. Moreover, if we obtain Fail, then we conclude that with probability at least $1-\delta$ the graph $G^{+W-\Delta}$ contains a negative cycle. This suggests the following idea. We make a call to TestScale$(G^{+W-\Delta}, \delta)$, and consider the two outcomes:

1. TestScale$(G^{+W-\Delta}, \delta) = \phi$. Then, we set $G := G_\phi$ and increase $\Delta := 2\Delta$.
2. TestScale$(G^{+W-\Delta}, \delta) =$ Fail. Then, we decrease $\Delta := \Delta/2$.

If we are in Case 1, then the minimum edge weight $-W'$ of G_ϕ is increased by Δ. This in turn, decreases the gap $W' - M^*$ (note that at all times $M^* \leq W'$). Thus, larger Δ implies larger progress in decreasing $W' - M^*$. This is why in this case we double Δ. On the other hand, if we are in Case 2 then by the guarantee of Lemma 8.30, we conclude that with probability at least $1-\delta$ the graph $G^{+W-\Delta}$ contains a negative cycle. Intuitively,

Algorithm 20 The fast implementation of THRESHOLD. Given a graph G, it computes the smallest integer $M^* \geq 0$ such that G^{+M^*} contains no negative cycle. See Lemma 8.32.

```
procedure FastThreshold($G$)
    Let $G_0 \leftarrow G$ and $\Delta_0 \leftarrow 2$
    $T \leftarrow \Theta(\log(nW))$ with sufficiently large hidden constant
    for $t \leftarrow 0, \dots, T-1$ do
        Let $-W_t$ be the smallest edge weight in $G_t$
        if $W_t \leq 24$ then
            for $j \leftarrow 23, \dots, 1, 0$ do
                if FindNegCycle($G_t^{+j}$) = NegCycle then return $j+1$
        if TestScale($G_t^{+W_t-\Delta_t}, 0.01$) = $\phi$ then
            $G_{t+1} \leftarrow (G_t)_\phi,\ \Delta_{t+1} \leftarrow 2 \cdot \Delta_t$
        else
            $G_{t+1} \leftarrow G_t,\ \Delta_{t+1} \leftarrow \max\{1, \frac{\Delta_t}{2}\}$
    return $W_T$
```

this means that Δ is too large. Therefore, we halve Δ to eventually make progress in Case 1 again.

In short, we know that when $G^{+W-\Delta}$ does not have negative cycles, or equivalently $W - M^* \geq \Delta$, then with probability at least $1 - \delta$ we will make progress in Case 1 by decreasing the gap $W - M^*$. On the other hand, if we are in Case 2 and $G^{+W-\Delta}$ has a negative cycle, or equivalently $W - M^* < \Delta$, then we will make progress by decreasing Δ.

Perhaps surprisingly, we will show that this idea can be implemented by choosing $\delta = 0.01$, and not $1/\mathrm{poly}(n)$ as in the implementation of Lemma 8.31 (which was the reason for getting an extra $O(\log n)$-factor there). For this, we will formalize the progress as some *drift function* that decreases in expectation in each iteration, and then apply a *drift theorem* (see Theorem 8.34).

The Algorithm Now we formalize this approach. We proceed in an iterative way. At iteration t, we have a graph G_t with minimum weight $-W_t$, and we maintain a parameter Δ_t. We make a call to SCALE($G^{+W_t-\Delta_t}, \delta$) with $\delta := 0.01$. If we obtain a potential ϕ as answer, we set $G_{t+1} := (G_t)_\phi$ and $\Delta_{t+1} := 2\Delta$. Otherwise, we set $G_{t+1} := G_t$ and $\Delta_{t+1} := \frac{1}{2}\Delta_t$. After $T = \Theta(\log(nW))$ iterations, we stop and return W_T as the answer. The complete pseudocode (which additionally handles some corner cases) is in Algorithm 20.

To quantify the progress made by the algorithm, we define the following *drift function* at iteration t:

$$D_t := (W_t - M^*)^{20} \cdot \max\left\{\frac{2\Delta_t}{W_t - M^*}, \frac{W_t - M^*}{2\Delta_t}\right\}, \tag{8.1}$$

Observe that we always have $\Delta_t \geq 1$ and $W_t \geq M^*$ throughout the algorithm. To cover the case $W_t = M^*$ (where the above expression leads to a division by 0), formally we actually define the drift function by

$$D_t := \max\left\{(W_t - M^*)^{19} \cdot 2\Delta_t, \frac{(W_t - M^*)^{21}}{2\Delta_t}\right\}. \tag{8.2}$$

For the sake of readability, in the following we work with (8.1), with the understanding that formally we mean (8.2).

We will show that D_t decreases by a constant factor (in expectation) in each iteration of the for-loop in Line 4. Note that when D_t reaches 0, then we have that $W_t = M^*$, so we are done.

Lemma 8.33 (Negative Drift)**.** *For any $d > 0$ and $t \geq 0$ it holds that*

$$\mathbb{E}(D_{t+1} \mid D_t = d) \leq 0.7 \cdot d.$$

Before proving Lemma 8.33, let us see how to obtain Lemma 8.32 from it. For this, we will use the following tool:

Theorem 8.34 (Multiplicative Drift, see e.g. [Len17, Theorem 18])**.** *Let $(X_t)_{t\geq 0}$ be a sequence of nonnegative random variables with a finite state space $\mathcal{S}$ of nonnegative integers. Suppose that $X_0 = s_0$, and there exists $\delta > 0$ such that for all $s \in \mathcal{S} \setminus \{0\}$ and all $t \geq 0$, $\mathbb{E}(X_{t+1} \mid X_t = s) \leq (1-\delta)s$. Then, for all $r \geq 0$,*

$$\mathbb{P}(X_r > 0) \leq e^{-\delta \cdot r} \cdot s_0.$$

Proof. By Markov's inequality, $\mathbb{P}(X_r > 0) = \mathbb{P}(X_r \geq 1) \leq \mathbb{E}(X_r)$. By applying the bound $\mathbb{E}(X_{t+1} \mid X_t = d) \leq (1-\delta)d$ for r times, we obtain that

$$\mathbb{P}(X_r > 0) \leq (1-\delta)^r \cdot s_0 \leq \exp(-\delta r) \cdot s_0. \qquad \square$$

Proof of Lemma 8.32. See Algorithm 20 for the pseudocode. First we analyze the running time. During each iteration of the for-loop, it either holds that $W_t \leq 24$ and we solve the problem directly using at most 24 calls to DetectNegCycle, or we make a call to TestScale. Each call to TestScale takes time $O(T_{\text{RSSSP}}(m,n))$ by Lemma 8.30, and we only make the calls to DetectNegCycle once which take total time $O(T_{\text{RSSSP}}(m,n)\log n)$ by Corollary 8.24. Since $T = \Theta(\log(nW))$, the overall running time is bounded by $O(T_{\text{RSSSP}}(m,n)\log n + T_{\text{RSSSP}}(m,n)\log(nW))$, as claimed.

Now we analyze correctness. Note that at every iteration, G_t is equivalent to G since the only way we modify the graph is by adding potentials (see Lemma 7.16). Thus, if at some point we have that $W_t \leq 24$ then the correct answer lies in the range $0 \leq M^* \leq 24$. The for-loop in Line 7 exhaustively checks which is the correct value by making calls to DetectNegCycle. By Corollary 8.24, this is correct with high probability.

Now suppose the algorithm does not terminate in Line 8. We claim that the final drift D_T is zero with high probability. Note that this implies correctness, since $D_T = 0$ if and only if $W_T = M^*$ (to see this, observe that $\Delta_T \geq 1$ due to Line 12). To prove the claim, we will use Theorem 8.34. Note that Lemma 8.33 gives us that $\mathbb{E}(D_{t+1} \mid D_t = d) \leq 0.7d$. Moreover, we can bound the initial drift D_0 as

$$D_0 = (W - M^*)^{20} \cdot \max\left\{\frac{2\Delta_0}{W - M^*}, \frac{W - M^*}{2\Delta_0}\right\} \leq (W - M^*)^{21} \cdot 2\Delta_0 \leq 4W^{21}.$$

Hence, Theorem 8.34 yields that $\mathbb{P}(D_T > 0) \leq \exp(-0.7T) \cdot 4W^{21}$. Since $T = \Theta(\log(nW))$, we conclude that $\mathbb{P}(D_T > 0) \leq n^{-\Omega(1)}$, which finishes the proof. ◻

Proof of Lemma 8.33. Focus on iteration t of the for-loop in Line 4. Let E_1 be the event that we obtain a potential ϕ (i.e. that the if-statement in Line 9 succeeds) and let $E_2 := \neg E_1$ be the complement. We start by observing how the parameters W_{t+1} and Δ_{t+1} change depending on whether E_1 or E_2 occur.

▷ Claim 8.35. If E_1 occurs, then $W_{t+1} \leq W_t - \frac{\Delta_t}{4}$, and $\Delta_{t+1} = 2\Delta_t$.

Proof. If the call to TestScale in Line 9 returns a potential ϕ, then we set $G_{t+1} = (G_t)_\phi$ and $\Delta_{t+1} = 2\Delta_t$. Observe that the minimum weight of $G_t^{+W_t-\Delta_t}$ is Δ_t. Hence, Lemma 8.30 guarantees that the minimum weight of $(G_t)_\phi^{+W_t-\Delta_t}$ is at least $-\frac{3}{4}\Delta_t$. Since $G_{t+1} = (G_t)_\phi$ is defined by substracting $W_t - \Delta_t$ from every edge weight in $(G_t)_\phi^{+W_t-\Delta_t}$, we obtain that $-W_{t+1} \geq -W_t + \frac{1}{4}\Delta_t$. ◁

▷ Claim 8.36. If E_2 occurs, then $W_{t+1} = W_t$ and $\Delta_{t+1} = \max\{1, \Delta_t/2\}$ and $D_{t+1} \leq 2D_t$.

Proof. The first two statements are immediate by Line 12. Towards the third statement, for the function $f(x) := \max\{x, 1/x\}$ we observe that if $x, y > 0$ differ by at most a factor 2 then also $f(x), f(y)$ differ by at most a factor 2. Now we use that $D_t = (W_t - M^*)^{20} \cdot f(2\Delta_t/(W_t - M^*))$. Since $\Delta_t \geq 1$, it holds that Δ_t, Δ_{t+1} differ by at most a factor 2, and thus D_t, D_{t+1} differ by at most a factor 2. ◁

With these claims, we proceed to bound the drift D_{t+1} when $D_t > 0$. Recall that we defined

$$D_t = (W_t - M^*)^{20} \cdot \max\left\{\frac{2\Delta_t}{W_t - M^*}, \frac{W_t - M^*}{2\Delta_t}\right\}. \tag{8.1}$$

Note that it always holds that $W_t \geq M^*$ and $W_{t+1} \geq M^*$. Moreover, since $D_t > 0$, we can assume that $W_t - M^* > 0$, since otherwise $W_t - M^* = 0$ and hence $D_t = 0$. We proceed making a case distinction based on the term that achieves the maximum in (8.1).

Case 1 $\Delta_t \geq \frac{1}{2}(W_t - M^*)$: Then, we have that $D_t = (W_t - M^*)^{19} \cdot 2\Delta_t$. If E_1 occurs, then by Claim 8.35 it holds that $\Delta_{t+1} \geq \Delta_t \geq \frac{1}{2}(W_t - M^*) \geq \frac{1}{2}(W_{t+1} - M^*)$. Therefore, using (8.1) we can bound the drift D_{t+1} by

$$\begin{aligned} D_{t+1} &= (W_{t+1} - M^*)^{19} \cdot 2\Delta_{t+1} \\ &\leq (W_t - M^* - \tfrac{\Delta_t}{4})^{19} \cdot 4\Delta_t \\ &\leq (W_t - M^* - \tfrac{1}{8}(W_t - M^*))^{19} \cdot 4\Delta_t \\ &\leq (\tfrac{7}{8})^{19} \cdot 2D_t \leq 0.16D_t, \end{aligned}$$

where we used Claim 8.35 in the first inequality, and the second inequality follows since by the assumption of Case 1 we have that $\frac{\Delta_t}{4} \geq \frac{1}{8}(W_t - M^*)$.

If E_2 occurs instead, we make a further case distinction:

Case 1.1 $\Delta_t > W_t - M^*$: Note that if $\Delta_t = 1$, then since W_t and M^* are integers it follows that $W_t = M^*$, and consequently $D_t = 0$, which contradicts the assumption that $D_t > 0$. Therefore, we can assume that $\Delta_t \geq 2$. In particular, by Claim 8.36 we have $\Delta_{t+1} = \frac{1}{2}\Delta_t > \frac{1}{2}(W_t - M^*) = \frac{1}{2}(W_{t+1} - M^*)$. Thus, by (8.1) we can express the drift D_{t+1} as

$$D_{t+1} = (W_{t+1} - M^*)^{19} \cdot 2\Delta_{t+1} = (W_t - M^*)^{19} \cdot \Delta_t = \frac{D_t}{2}.$$

Case 1.2 $\Delta_t \leq W_t - M^*$: Observe that in this case $G^{+W_t - \Delta_t}$ contains no negative cycle. Moreover, we can assume that $W_t > 24$ since otherwise the problem is solved directly in Line 7. Therefore, by Lemma 8.30 we have that $\mathbb{P}(E_2) \leq 0.01$. Moreover, by by Claim 8.36 we have $D_{t+1} \leq 2D_t$.

Combining the above, we conclude that for Case 1 it holds that

$$\begin{aligned}\mathbb{E}(D_{t+1} \mid D_t) &\leq \mathbb{P}(E_1)\,\mathbb{E}(D_{t+1} \mid D_t, E_1) + \mathbb{P}(E_2)\,\mathbb{E}(D_{t+1} \mid D_t, E_2) \\ &\leq 1 \cdot 0.16 D_t + \max\left\{1 \cdot \tfrac{1}{2} D_t, 0.01 \cdot 2D_t\right\} \leq 0.66 D_t.\end{aligned}$$

Case 2 $\Delta_t < \frac{1}{2}(W_t - M^*)$: Then, it holds that $D_t = (W_t - M^*)^{21}/(2\Delta_t)$. If E_2 occurs, then by the same argument as in Case 1.2 we have that $D_{t+1} \leq 2D_t$ and $\mathbb{P}(E_2) \leq 0.01$.

If E_1 occurs instead, then we make a further case distinction:

Case 2.1 $\Delta_{t+1} < \frac{1}{2}(W_{t+1} - M^*)$: Then using (8.1), it holds that

$$D_{t+1} = \frac{(W_{t+1} - M^*)^{21}}{2\Delta_{t+1}} \leq \frac{(W_t - M^*)^{21}}{4\Delta_t} = \frac{D_t}{2},$$

where the inequality holds due to Claim 8.35.

Case 2.2 $\Delta_{t+1} \geq \frac{1}{2}(W_{t+1} - M^*)$: Then it holds that $D_{t+1} = (W_{t+1} - M^*)^{19} \cdot 2\Delta_{t+1}$. Since by the assumption of Case 2 we have $(W_t - M^*)/(2\Delta_t) \geq 1$ and by Claim 8.35 we have $\Delta_{t+1} = 2\Delta_t$, we can bound D_{t+1} as

$$\begin{aligned}D_{t+1} &= (W_{t+1} - M^*)^{19} \cdot 2\Delta_{t+1} \\ &\leq (W_{t+1} - M^*)^{19} \cdot 4\Delta_t \cdot \left(\frac{W_t - M^*}{2\Delta_t}\right)^2 \\ &= (W_{t+1} - M^*)^{19} \cdot (W_t - M^*)^2 \cdot \tfrac{1}{\Delta_t}.\end{aligned} \tag{8.3}$$

By Claim 8.35, we have that $W_{t+1} \leq W_t - \frac{\Delta_t}{4}$. Hence, can bound $W_{t+1} - M^*$ as

$$\begin{aligned} W_{t+1} - M^* &= \tfrac{16}{17}(W_{t+1} - M^*) + \tfrac{1}{17}(W_{t+1} - M^*) \\ &\leq \tfrac{16}{17}(W_t - M^* - \tfrac{\Delta_t}{4}) + \tfrac{1}{17}(W_{t+1} - M^*) \\ &= \tfrac{16}{17}(W_t - M^*) - \tfrac{16}{17} \cdot \tfrac{\Delta_t}{4} + \tfrac{1}{17}(W_{t+1} - M^*). \end{aligned} \tag{8.4}$$

By Claim 8.35 and the assumption of Case 2.2, we have that $2\Delta_t = \Delta_{t+1} \geq \frac{1}{2}(W_{t+1} - M^*)$. This implies that $\frac{\Delta_t}{4} \geq \frac{1}{16}(W_{t+1} - M^*)$. Plugging this into (8.4), we obtain that

$$\begin{aligned} W_{t+1} - M^* &\leq \tfrac{16}{17}(W_t - M^*) - \tfrac{1}{17}(W_{t+1} - M^*) + \tfrac{1}{17}(W_{t+1} - M^*) \\ &= \tfrac{16}{17}(W_t - M^*). \end{aligned} \tag{8.5}$$

Finally, we combine (8.3) and (8.5) to obtain that

$$\begin{aligned} D_{t+1} &\leq (W_{t+1} - M^*)^{19}(W_t - M^*)^2 \cdot \tfrac{1}{\Delta_t} \\ &\leq (\tfrac{16}{17})^{19}(W_t - M^*)^{21} \cdot \tfrac{1}{\Delta_t} \\ &= (\tfrac{16}{17})^{19} \cdot 2 \cdot D_t \\ &\leq 0.65 D_t \end{aligned}$$

Combining the subcases considered, we conclude that for Case 2 it holds that

$$\begin{aligned} \mathbb{E}(D_{t+1} \mid D_t) &\leq \mathbb{P}(E_1)\,\mathbb{E}(D_{t+1} \mid D_t, E_1) + \mathbb{P}(E_2)\,\mathbb{E}(D_{t+1} \mid D_t, E_2) \\ &\leq 1 \cdot \max\left\{\tfrac{1}{2}D_t, 0.65D_t\right\} + 0.01 \cdot 2D_t \leq 0.67 \cdot D_t. \end{aligned}$$

Since cases 1 and 2 are exhaustive, the proof is concluded. □

8.3.4 Putting Everything Together

Now we put the pieces together to prove our main theorem.

Main Theorem 7.1 (Negative-Weight SSSP)**.** *There is a Las Vegas algorithm which, given a directed graph G and a source node s, either computes a shortest path tree from s or finds a negative cycle in G, running in time $O((m + n \log\log n) \log^2 n \log(nW))$ with high probability (and in expectation).*

Proof. The algorithm alternatingly runs the following two steps, and interupts each step after it exceeds a time budget of $O((m + n \log\log n) \log^2 n \log(nW))$:

1. Run SSSP(G, s). If this algorithm finishes in time and returns a shortest path tree, we check that the shortest path tree is correct (by relaxing all edges and testing whether any distance in the tree changes) and return this shortest path tree in the positive case. Otherwise, we continue with step 2.

2. Run FindNegCycle(G) (using Lemma 8.32 to implement Threshold). If this algorithm finishes in time and returns a negative cycle, we verify that the output is indeed a negative cycle and return this negative cycle in the positive case. Otherwise, we continue with step 1.

The algorithm is clearly correct: Whenever it terminates, it reports a correct solution. Let us focus on the running time. We distinguish two cases: First, assume that G does *not* contain a negative cycle. By Theorem 8.19 step 1 runs in time $O((m + n \log\log n)\log^2 n \log(nW))$ with high probability and is not interrupted in this case. Moreover, the SSSP algorithm returns a correct shortest path tree with high probability, and thereby terminates the algorithm after just one iteration of step 1.

On the other hand, suppose that G contains a negative cycle. The algorithm runs step 1 which is wasted effort in this case, but costs only time $O((m+n \log\log n)\log^2 n \log(nW))$. Afterwards, by Lemmas 8.26 and 8.32, a single execution of step 2 runs within the time budget with high probability. Moreover, since Lemma 8.26 is a Las Vegas algorithm, it returns a true negative cycle and the algorithm terminates.

The previous two paragraphs prove that the algorithm terminates after successively running step 1 and step 2 in time $O((m + n \log\log n)\log^2 n \log(nW))$ with high probability. Since we independently repeat these steps until the algorithm terminates, the same bound applies to the expected running time. □

Next, we prove Theorem 7.2 using the previous Main Theorem 7.1 as a black-box.

Theorem 7.2 (Negative-Weight Single-Source Distances). *There is a Las Vegas algorithm, which, given a directed graph G and a source $s \in V(G)$, computes the distances from s to all other vertices in the graph (these distances are possibly $-\infty$ or ∞), running in time $O((m + n \log\log n)\log^2 n \log(nW))$ with high probability (and in expectation).*

Proof. First, remove all vertices from the graph not reachable from s and return distance ∞ for each such vertex. Then compute the set of strongly connected components $C_1, \dots, C_\ell$ in G in time $O(m + n)$. For every SCC C_i, run our SSSP algorithm from Main Theorem 7.1 on $G[C_i]$ to detect whether it contains a negative cycle. For every vertex contained in a SCC with a negative cycle, we return distance $-\infty$ (as this SCC is reachable from s and contains a negative cycle, we can loop indefinitely). Similarly, report $-\infty$ for all vertices reachable from one of the $-\infty$-distance vertices. After removing all vertices at distance $-\infty$, the remaining graph does no longer contain a negative cycle. We may therefore run the SSSP algorithm on the remaining graph to compute the missing distances.

Let n_i and m_i denote the number of vertices and edges in the subgraph $G[C_i]$. Then the total running time is

$$\begin{aligned}
O\Bigg(T_{\mathrm{SSSP}}(m, n, W) + \sum_i T_{\mathrm{SSSP}}(m_i, n_i, W)\Bigg)& \\
&= O\left(\left(m + n \log\log n + \sum_i m_i + \sum_i n_i \log\log n\right) \log n^2 \log(nW)\right) \\
&= O((m + n \log\log n) \log^2 n \log(nW)),
\end{aligned}$$

using that $\sum_i m_i \leq m$ and that $\sum_i n_i \leq n$. $\square$

9 Further Results

In this chapter we present the following two side results: In Section 9.1 we present our algorithm to compute the minimum cycle mean, proving Theorem 7.3. In Section 9.2 we present our strong directed Low-Diameter Decomposition, establishing Theorem 7.5.

9.1 Minimum Cycle Mean

Recall that in a directed graph G, the mean of a cycle C is defined as $\bar{w}(C) = w(C)/|C|$. In this section we present our algorithm to computes the minimum cycle mean of a graph G:

Theorem 7.3 (Minimum Cycle Mean)**.** *There is a Las Vegas algorithm, which given a directed graph G finds a cycle C with minimum mean weight $\bar{w}(C) = \min_{C'} \bar{w}(C')$, running in time $O((m + n \log\log n) \log^2 n \log(nW))$ with high probability (and in expectation).*

Given a directed graph G, we denote by $\mu^*(G)$ the value of the minimum cycle mean, i.e., $\mu^*(G) := \min_C \bar{w}(C)$. To develop our algorithm, the following characterization of the minimum cycle mean will be useful:

Lemma 9.1. *Let G be a directed graph. Then,*

$$\mu^*(G) = -\min\{Q \in \mathbb{Q} \mid G^{+Q} \textit{ contains no negative cycle}\}.$$

Proof. By definition, we have that $\mu^*(G) = \min_C \bar{w}(C)$. Equivalently, $\mu^*(G)$ is the largest rational number μ such that $\mu \leq w(C)/|C|$ holds for all cycles C in G. In particular, $w(C) - \mu \cdot |C| \geq 0$ holds for all cycles C, which is equivalent to $G^{-\mu}$ not having negative cycles. □

Recall that Threshold(G), computes the minimum integer $M^* \geq 0$ such that G^{+M^*} contains no negative cycle (Definition 8.25). This is very similar to the characterization of the minimum cycle mean given by Lemma 9.1, except that the latter minimizes over rational numbers that are not necessarily nonnegative. To overcome this, we will use the following simple propositions:

Proposition 9.2. *Let G be a directed graph and let $a \geq 1, b \geq 0$ be integers. Let H be a copy of G where each edge has weight $w_H(e) := a \cdot w_G(e) + b$. Let C be any cycle in G. Then, $\bar{w}_H(C) = a \cdot \bar{w}_G(G) + b$.*

Proof. Note that the weight of C in H is exactly $w_H(C) = a \cdot w_G(C) + b \cdot |C|$. Therefore, the cycle mean of C in H equals $\bar{w}_H(C) = a \cdot w_G(C)/|C| + b = a \cdot \bar{w}_G(C) + b$. □

Proposition 9.3. *Let C and C' be two cycles in a directed graph G with distinct means, i.e. $\bar{w}(C) \neq \bar{w}(C')$. Then, $|\bar{w}(C) - \bar{w}(C')| \geq 1/n^2$.*

Proof. By definition, we can express $|\bar{w}(C) - \bar{w}(C')|$ as

$$\left|\frac{w(C)}{|C|} - \frac{w(C')}{|C'|}\right| = \left|\frac{w(C)|C'| - w(C')|C|}{|C| \cdot |C'|}\right| \geq \frac{1}{|C||C'|},$$

where we used that $\bar{w}(C) \neq \bar{w}(C')$. Since $|C|, |C'| \leq n$, we have that this is at least $1/n^2$. □

We will use the following lemma, which is a Las Vegas implementation of Lemma 8.32.

Lemma 9.4. *Let G be a directed graph. There is a Las Vegas algorithm which computes $\textsc{Threshold}(G)$ (see Definition 8.25) and runs in time $O((m + n \log\log n) \log^2 n \log(nW))$ with high probability (and in expectation).*

Proof. The algorithm computes $M^* = \textsc{Threshold}(G)$ using Lemma 8.32. By definition, this returns the smallest integer M^* such that G^{+M^*} contains no negative cycles with high probability (recall Definition 8.25). To turn it into a Las Vegas algorithm, we need to verify that the output is correct. For this, we add a source vertex s connected with 0-weight edges to all other vertices and use Main Theorem 7.1 to test if G^{+M^*} contains no negative cycles and that G^{+M^*-1} contains negative cycles. If either test fails, the algorithm restarts.

The correctness of this procedure follows since Main Theorem 7.1 is a Las Vegas algorithm. For the running time, observe that call to Lemma 8.32 (using the bound on $T_{\text{RSSSP}}(m, n)$ given by Theorem 8.1) and the calls to Main Theorem 7.1 run in time

$$O((m + n \log\log n) \log^2 n \log(nW)).$$

Moreover, Lemma 8.32 guarantees that the value M^* is correct with high probability. Thus, the algorithm terminates in time

$$O((m + n \log\log n) \log^2 n \log(nW))$$

with high probability. □

Proof of Theorem 7.3. We construct a graph H by copying G and modifying each edge weight to $n^2 w(e) - n^3 L$, where L is the largest edge-weight in G. Then, we compute $M^* := \textsc{Threshold}(H)$ using Lemma 9.4. Finally, we find a negative cycle in H^{+M^*-1} using Lemma 8.26. See Algorithm 21 for the pseudocode.

The running time is dominated by the calls to $\textsc{Threshold}$ and $\textsc{FindNegCycle}$. Using Lemma 9.4 the call to $\textsc{Threshold}$ takes time $O((m + n \log\log n) \log^2 n \log(nW))$

Algorithm 21 Given a graph G the procedure returns cycle C of minimum mean weight with high probability. See Theorem 7.3.

```
1 procedure MinCycleMean($G$)
2     Let $L$ be the largest weight in $G$
3     Let $H$ be a copy of $G$ with edge weights $w_H(e) \leftarrow n^2 \cdot w_G(e) - n^3 L$.
4     Compute $M^* \leftarrow$ Threshold($H$) using Lemma 9.4
5     $C \leftarrow$ FindNegCycle($H^{+M^*-1}$)
6     return $C$
```

with high probability. By Lemma 8.26 the call to FindNegCycle, (using Lemma 8.32 to implement Threshold and Theorem 8.1 to bound $T_{\mathrm{RSSSP}}(m,n)$) takes time $O((m + n \log\log n)\log^2 n \log(nW))$ with high probability as well. Thus, the algorithm runs in the claimed running time.

To analyze the correctness, note that Proposition 9.2 implies that a cycle C is the minimizer of $\bar{w}_G(C)$ if and only if it is the minimizer of $\bar{w}_H(C)$. Thus, it suffices to find a cycle of minimum mean in H. We will argue that the cycle found by the algorithm is the minimizer.

▷ Claim 9.5. The value M^* computed in Line 4 satisfies $M^* = \lceil -\mu^*(H) \rceil$.

Proof. We observe that the minimum cycle mean in H is non-positive, i.e., $\mu^*(H) \leq 0$. To see this, note that any cycle C in G has weight at most $w_G(C) \leq nL$. Thus, by the way we set the weights in H, any cycle in H has weight $w_H(C) = n^2 w_G(C) - n^3 L|C| \leq n^3 L - n^3 L = 0$. This means that in Lemma 9.1 we can minimize over $Q \geq 0$, i.e. that

$$\mu^*(H) = -\min\{0 \leq Q \in \mathbb{Q} \mid H^{+Q} \text{ contains no negative cycle}\}. \tag{9.1}$$

Recall that by definition of Threshold(H), M^* is the smallest nonnegative integer such that H^{+M^*} has no negative cycles, i.e.

$$M^* = \min\{0 \leq M \in \mathbb{Z} \mid H^{+M} \text{ contains no negative cycle}\}. \tag{9.2}$$

Combining (9.1) and (9.2), we conclude that $M^* = \lceil -\mu^*(H) \rceil$, as claimed. ◁

It follows that H^{+M^*-1} indeed contains a negative cycle. By Lemma 9.4, the call to Threshold is correct. Hence, H^{+M^*-1} contains a negative cycle and the call to FindNegCycle is correct by Lemma 8.26. Let C be the cycle obtained in Line 5. Since it has negative weight in H^{+M^*-1}, its weight in H is less than $-|C|(M^* - 1)$. Hence, it holds that $\bar{w}_H(C) < -M^* + 1$. Moreover, since H^{+M^*} contains no negative cycle, every cycle C' has mean weight $\bar{w}_H(C') \geq -M^*$.

Now consider a minimum mean cycle C'. As we have seen, we have

$$-M^* \leq \bar{w}_H(C') \leq \bar{w}_H(C) < -M^* + 1. \tag{9.3}$$

Assume for the sake of contradiction that $\bar{w}_H(C) \neq \bar{w}_H(C')$. Then by Proposition 9.3 we have that $|\bar{w}_G(C) - \bar{w}_G(C')| \geq 1/n^2$, and by Proposition 9.2 it holds that $\bar{w}_H(C) = n^2 \cdot w_G(C) - n^3L$ and $\bar{w}_H(C') = n^2 \cdot w_G(C') - n^3L$. Combining these facts, we obtain that $|\bar{w}_H(C) - \bar{w}_H(C')| \geq 1$. This contradicts Equation (9.3). Hence, we obtain $\bar{w}_H(C) = \bar{w}_H(C')$, so the computed cycle C is a minimizer of $\bar{w}_H(C)$ and thus also of $\bar{w}_G(C)$. □

9.2 Low-Diameter Decomposition

In this section we establish our strong Low-Diameter Decomposition (LDD). Recall that in a strong LDD (as defined in Definition 7.4), the goal is to select a small set of edges S such that after removing the edges in S, each strongly connected component in the remaining graph has bounded diameter. Our result is the following theorem, which proves that strong LDDs exist (which was known by [BNW22]) and can be efficiently computed (which was open):

Theorem 7.5 (Strong Low-Diameter Decomposition). *There is a strong Low-Diameter Decomposition with overhead $O(\log^3 n)$, computable in time $O((m + n \log\log n) \log^2 n)$ with high probability (and in expectation).*

9.2.1 Heavy and Light Vertices

In the algorithm we will distinguish between *heavy* and *light* vertices, depending on how large the out- and in-balls of these vertices are. To classify vertices as heavy or light, we rely on the following simple lemmas:

Lemma 9.6 (Estimate Ball Sizes). *Let $\varepsilon > 0$. Given a directed graph G with nonnegative edge weights and $r > 0$, we can approximate $|B^{out}(v, r)|$ with additive error εn for each vertex v. With high probability, the algorithm succeeds and runs in time $O(\varepsilon^{-2} \log n \cdot (m + n \log\log n))$.*

Proof. Sample random vertices $v_1, \dots, v_k \in V(G)$ (with repetition) for $k := 5\varepsilon^{-2} \log n$. Compute $B^{in}(v_i, r)$ for all $i \in [k]$. Using Dijkstra's algorithm with Thorup's priority queue [Dij59; Tho04], this step runs in time $O(k \cdot (m + n \log\log n)) = O(\varepsilon^{-2} \log n \cdot (m + n \log\log n))$. Now return for each vertex v, the estimate

$$b(v) := \frac{n}{k} \cdot |\{\, i \in [k] \mid v \in B^{in}(v_i, r) \,\}|.$$

We claim that this estimate is accurate. Let I_i denote the indicator variable whether $v_i \in B^{out}(v, r)$, and let $I := \sum_{i=1}^{k} I_i$. Then the random variable $b(v)$ is exactly

$$b(v) = \frac{n}{k} \cdot I.$$

Note that $\mathbb{P}(I_i = 1) = |B^{out}(v, r)|/n$. Therefore, in expectation we have

$$\mathbb{E}(b(v)) = \frac{n}{k} \cdot \sum_{i=1}^{k} \mathbb{P}(I_i = 1) = \frac{n}{k} \cdot \frac{k}{n} \cdot |B^{out}(v, r)| = |B^{out}(v, r)|.$$

Using Chernoff's bound we have $\mathbb{P}(|I - \mathbb{E}(I)| > a) < 2\exp(-2a^2/k)$. For $a := \varepsilon k$ we obtain

$$\mathbb{P}(|b(v) - \mathbb{E}(b(v))| > \varepsilon n) = \mathbb{P}(|I - \mathbb{E}(I)| > \varepsilon k) < 2\exp(-2\varepsilon^2 k) \leq 2n^{-10}.$$

Hence, with high probability the computed estimates are accurate. □

Lemma 9.7 (Heavy/Light Classification)**.** *There is an algorithm* Light(G, r) *that, given a directed graph G and a radius r, returns a set $L \subseteq V(G)$ with the following properties:*

- *For all $v \in L$, it holds that $|B^{out}(v, r)| \leq \frac{7}{8}n$.*
- *For all $v \in V(G) \setminus L$, it holds that $|B^{out}(v, r)| \geq \frac{3}{4}n$.*
- Light(G, r) *runs in time $O((m + n \log\log n) \log n)$.*

Proof. We run the previous Lemma 9.6 with parameter $\varepsilon := \frac{1}{16}$, and let L be the subset of vertices with estimated ball sizes at most $\frac{13}{16}n$. With high probability, the estimates have additive error at most $\varepsilon n = \frac{1}{16}n$. Therefore any vertex $v \in L$ satisfies $|B^{out}(v, r)| \leq \frac{13}{16}n + \frac{1}{16}n = \frac{7}{8}n$ and any vertex $v \in V(G) \setminus L$ satisfies $|B^{out}(v, r)| \geq \frac{13}{16}n - \frac{1}{16}n = \frac{3}{4}n$. The running time is dominated by Lemma 9.6 which runs in time $O((m + n \log\log n) \log n)$ as claimed. □

9.2.2 The Strong Low-Diameter Decomposition

The strong LDD works as follows: Let $R = \frac{D}{10 \log n}$. First, we run Lemma 9.7 on G with radius R to compute a set L^{out} and we run Lemma 9.7 on the reversed graph with radius R to compute a set L^{in}. We refer to the vertices in L^{out} as *out-light*, to the vertices in L^{in} as *in-light*, and to the vertices in $V(G) \setminus (L^{out} \cup L^{in})$ as *heavy*. Then we distinguish two cases:

- *The heavy case:* If there is a heavy vertex $v \in V(G) \setminus (L^{out} \cup L^{in})$, we compute the set of vertices W that both reach v and are reachable from v within distance R, i.e., $W = B^{out}(v, R) \cap B^{in}(v, R)$. Let T^{out}, T^{in} denote the shortest path trees from v to W and from W to v, respectively. Let C be the union of vertices in T^{out} and T^{in}. We *collapse* C (that is, we replace all vertices in C by a single super-vertex) and consider the remaining (multi-)graph G/C. We recursively compute the strong LDD in G/C, resulting in a set of edges S. In S we uncollapse all edges involving the super-vertex (i.e., for any edge $(v, u) \in E(G)$ which became an edge (C, u) in the collapsed graph, we revert (C, u) back to (v, u)) and return S.

Algorithm 22 The implementation of the strong Low-Diameter Decomposition (see Theorem 7.5) that either returns a set of edges $S \subseteq E(G)$ or FAIL.

```
procedure StrongLDD($G, D$)
    if $n \leq 100$ then return $E(G)$
    Let $R := \frac{D}{10 \log n}$
    Compute $L^{out} \leftarrow$ Light($G, R$)
    Compute $L^{in} \leftarrow$ Light($G^{rev}, R$) (here, $G^{rev}$ is the graph $G$ with reversed edges)
    (The heavy case)
    if there is a heavy vertex $v \in V(G) \setminus (L^{out} \cup L^{in})$ then
        Let $W \leftarrow B^{out}(v, R) \cap B^{in}(v, R)$
        Compute shortest path trees $T^{out}$ from $v$ to $W$, and $T^{in}$ from $W$ to $v$
        Let $C$ be the union of vertices in $T^{out}, T^{in}$
        $S \leftarrow$ StrongLDD($G/C, D - 4R$)
        return $S$ after uncollapsing all edges
    (The light case)
    $S \leftarrow \emptyset$
    while there is $v \in V(G) \cap L^{out}$ do
        $r \sim \mathrm{Geom}(R^{-1} \cdot 10 \log n_0)$
        if $r > R$ then return Fail
        $S \leftarrow S \cup \partial B^{out}(v, r) \cup$ StrongLDD($G[B^{out}(v, r)], D$)
        $G \leftarrow G \setminus B^{out}(v, r)$
    while there is $v \in V(G) \cap L^{in}$ do
        $r \sim \mathrm{Geom}(R^{-1} \cdot 10 \log n_0)$
        if $r > R$ then return Fail
        $S \leftarrow S \cup \partial B^{in}(v, r) \cup$ StrongLDD($G[B^{in}(v, r)], D$)
        $G \leftarrow G \setminus B^{in}(v, r)$
    return $S$
```

- *The light case:* If there is no heavy vertex, then each vertex is out-light or in-light. For each vertex v (which is out-light, say) we can therefore proceed in the standard way: Sample a radius r from a geometric distribution with parameter $O(\log n/D)$, cut the edges leaving $B^{out}(v, r)$ and recur on both the inside and the outside of the ball $B^{out}(v, r)$.

We summarize the pseudocode with the precise parameters in Algorithm 22. Throughout this section, we denote by n_0 the size of the original graph and by n the size of the current graph G (in the current recursive call of the algorithm).

Lemma 9.8 (Strong Diameter of Algorithm 22). *With high probability,* StrongLDD(G, D) *either returns* Fail *or a set of edges $S \subseteq E(G)$ such that every strongly connected component C of $G \setminus S$ has diameter at most D, i.e.,* $\max_{u,v \in C} \mathrm{dist}_{G[C]}(u, v) \leq D$.

Proof. With high probability, the heavy-light classification works correctly in the execution of StrongLDD(G, D) (and all recursive calls). We condition on this event and treat the classification as perfect.

As before, we have to distinguish the heavy and the light case. In the heavy case, let v be the heavy vertex and let W, T^{out}, T^{in}, C be as in the algorithm. We claim that the induced subgraph $G[C]$ has diameter at most $4R$. Take any vertex $x \in C$; it suffices to prove that both $\mathrm{dist}_{G[C]}(v, x) \leq 2R$ and $\mathrm{dist}_{G[C]}(x, v) \leq 2R$. We show the former claim and omit the latter. There are two easy cases: Either we have $x \in T^{out}$ in which case we immediately have that $\mathrm{dist}_{G[C]}(v, x) \leq R$ (as any path in T^{out} has length at most R). Or we have $x \in T^{in}$, in which case there exists some intermediate vertex $y \in W$ with $\mathrm{dist}_{G[C]}(y, x) \leq R$. But then also $\mathrm{dist}_{G[C]}(v, y) \leq R$ and in combination we obtain $\mathrm{dist}_{G[C]}(v, x) \leq 2R$ as claimed.

Recall that the algorithm collapses the vertices in C, and computes a strong LDD S on the remaining multigraph with parameter $D - 4R$. We assume by induction that the recursive call computes a correct strong decomposition (for G/C). To see that the decomposition is also correct for G, take any two vertices u, v in the same strongly connected component in $G \setminus S$. We have that $\mathrm{dist}_{(G/C)\setminus S}(u, v) \leq D - 4R$. If the shortest u-v-path in G/C does not touch the supervertex, then we immediately have $\mathrm{dist}_{G\setminus S}(u, v) \leq D - 4R \leq D$. If the shortest path touches the supervertex, then we can replace the path through C by a path of length $\mathrm{diam}(G[C]) \leq 4R$. It follows that $\mathrm{dist}_{G\setminus S}(u, v) \leq D - 4R + 4R \leq D$.

The correctness of the light case is exactly as in the known LDD by [BNW22], and similar to Lemma 7.8: For every ball $B^{out}(v, r)$ (or $B^{in}(v, r)$) that the algorithm carves out, we remove all outgoing edges $\partial B^{out}(v, r)$ (or all incoming edges $\partial B^{in}(v, r)$, respectively). Thus, two vertices x, y in the remaining graph are part of the same strongly connected component only if both $x, y \in B^{out}(v, r)$ or both $x, y \notin B^{out}(v, r)$. The algorithm continues the loop on all vertices outside $B^{out}(v, r)$ and recurs inside $B^{out}(v, r)$. By induction, both calls succeed and reduce the diameter to at most D.

Eventually the algorithm reaches a base case where G contains only a constant number of nodes and edges—in this case, we can select S to be the whole set of edges. ◻

Lemma 9.9 (Sparse Hitting of Algorithm 22)**.** *For any edge $e \in E(G)$, the probability that e is contained in the output of* StrongLDD(G, D) *is at most* $O(\frac{w(e)}{D} \cdot \log^3(n_0) + \frac{1}{\mathrm{poly}(n)})$.

Proof. In this proof we condition on the event that the initially computed heavy/light classification is correct. Since this event happens with high probability, we only increase the hitting probabilities by $\frac{1}{\mathrm{poly}(n)}$ for all edges.

Let $p(n, w, D)$ be an upper bound on the probability that an edge of weight w is contained in the output of StrongLDD(G, D), where G is an n-vertex graph. We inductively prove that $p(n, w, D) \leq \frac{w}{D} \cdot 1000 \log(n_0) \log^2(n)$ which is as claimed. We distinguish the heavy and light case in Algorithm 22.

The Light Case. Suppose that the algorithm enters the light case (that is, there is no vertex classified as heavy). Focus on some edge $e = (x, y)$ of weight $w = w(e)$. We distinguish three cases for each iteration. Suppose that the current iteration selects an out-light vertex v.

- $x, y \notin B^{out}(v, r)$: The edge e is not touched in this iteration and remains a part of the graph G. It may or may not be included in the output, depending on the future iterations.
- $x \in B^{out}(v, r)$ and $y \notin B^{out}(v, r)$: In this case $e \in \partial B^{out}(v, r)$ and thus the edge is included into S.
- $y \in B^{out}(v, r)$: The edge is not included in $\partial B^{out}(v, r)$. It may however be included in the recursive call on $B^{out}(v, r)$. In the recursive call we have that $|B^{out}(v, r)| \leq |B^{out}(v, R)| \leq \frac{7n}{8}$, as $r \leq R$ (in the opposite case the algorithm fails and no edge is returned) and by Lemma 9.7 as v is out-light.

Combining these cases, we obtain the following recursion for $p(n, w, D)$. In the calculation we abbreviate $q := R^{-1} \cdot 10 \log(n_0)$:

$$\begin{aligned} p(n, w, D) &\leq \max_{v \in V(G)} \mathop{\mathbb{P}}_{r \sim \mathrm{Geom}(q)} (y \notin B^{out}(v, r) \mid x \in B^{out}(v, r)) + p(\tfrac{7n}{8}, w, D) \\ &\leq \max_{v \in V(G)} \mathop{\mathbb{P}}_{r \sim \mathrm{Geom}(q)} (r < \mathrm{dist}(v, y) \mid r \geq \mathrm{dist}(v, x)) + p(\tfrac{7n}{8}, w, D) \\ &\leq \max_{v \in V(G)} \mathop{\mathbb{P}}_{r \sim \mathrm{Geom}(q)} (r < \mathrm{dist}(v, x) + w \mid r \geq \mathrm{dist}(v, x)) + p(\tfrac{7n}{8}, w, D) \end{aligned}$$

Let $r' := r - \mathrm{dist}(v, x)$. Conditioned on the event $r \geq \mathrm{dist}(v, x)$, r' is a nonnegative random variable and by the memoryless property of geometric distributions, r' is sampled from $\mathrm{Geom}(q)$, too:

$$\begin{aligned} &\leq \max_{v \in V(G)} \mathop{\mathbb{P}}_{r' \sim \mathrm{Geom}(q)} (r < w) + p(\tfrac{7n}{8}, w, D) \\ &\leq wq + p(\tfrac{7n}{8}, w, D) \\ &\leq \frac{w}{D} \cdot 100 \log(n_0) \log(n) + p(\tfrac{7n}{8}, w, D). \end{aligned}$$

In the last step, we have plugged in $q = R^{-1} \cdot 10 \log(n_0) = \frac{1}{D} \cdot 100 \log(n_0) \log(n)$. It follows by induction that $p(n, w, D) \leq \frac{w}{D} \cdot 100 \log(n_0) \log(n) \log_{8/7}(n) \leq \frac{w}{D} \cdot 1000 \log(n_0) \log^2(n)$.

The same analysis applies also to the in-balls with "B^{in}" in place of "B^{out}".

The Heavy Case. In the heavy case, the algorithm selects a heavy vertex v, computes the sets $W = B^{out}(v, R) \cap B^{in}(v, R)$ and $C \supseteq W$ and recurs on the graph G/C in which we contract the vertex set C to a single vertex. We have $|B^{out}(v, R)|, |B^{in}(v, R)| > \frac{3n}{4}$ by Lemma 9.7 since v is heavy. It follows that $|C| \geq |W| > \frac{n}{2}$ and therefore the contracted graph has size $|V(G/C)| \leq \frac{n}{2}$. As we call the algorithm recursively with parameter $D - 4R$ where $R = \frac{D}{10 \log n}$, we obtain the following recurrence:

$$p(n, w, D) \leq p(\tfrac{n}{2}, w, D - 4R).$$

Using the induction hypothesis, we obtain:

$$\begin{aligned}
p(n,w,D) &\leq \frac{w}{D-4R} \cdot 1000 \log^2(n_0) \log(\tfrac{n}{2}) \\
&\leq \frac{w}{D} \cdot \frac{1}{1 - \frac{4}{10\log n}} \cdot 1000 \log^2(n_0) \log(\tfrac{n}{2}) \\
&= \frac{w}{D} \cdot \frac{\log(n)}{\log(n) - \frac{4}{10}} \cdot 1000 \log^2(n_0) \cdot (\log(n) - 1) \\
&\leq \frac{w}{D} \cdot 1000 \log^2(n_0) \cdot \log(n).
\end{aligned}$$ □

Lemma 9.10 (Running Time of Algorithm 22)**.** *The algorithm* StrongLDD(G,D) *runs in time* $O((m + n_0 \log\log n_0)\log^2(n_0))$.

Proof. First focus on a single call of the algorithm and ignore the cost of recursive calls. It takes time $O((m + n_0 \log\log n_0)\log(n_0))$ to compute the heavy-light classification. In the heavy case, we can compute W, T^{out}, T^{in}, C in Dijkstra-time $O(m + n_0 \log\log n_0)$. In the light case, we can also carve out all balls $B^{out}(v,r)$ and $B^{in}(v,r)$ in total time $O(m + n_0 \log\log(n_0))$, although the formal analysis is more involved: Observe that we explore each vertex at most once spending time $O(\log\log n_0)$, and that we explore each edge at most once spending time $O(1)$. Since the analysis is similar to Lemma 7.8, we omit further details.

As the algorithm recurs on disjoint subgraphs of G, where the number of nodes in each subgraph is a constant factor smaller than the original number of nodes or less, the running time becomes $O((m + n_0 \log\log n_0)\log(n_0)^2)$. □

Lemma 9.11 (Failure Probability of Algorithm 22)**.** StrongLDD(G,D) *returns* Fail *with probability at most* $O(n_0^{-8})$.

Proof. As shown in detail in the previous lemmas, with every recursive call the number of vertices reduces by a constant factor and thus the recursion reaches depth at most $O(\log n_0)$. In each recursive call, the loops in Lines 13 and 18 run at most n_0 times. For each execution, the error event is that $r > R$, where $r \sim \mathrm{Geom}(R^{-1} \cdot 10\log(n_0))$. This event happens with probability at most $\exp(-10\log(n_0)) \leq n_0^{-10}$, and therefore the algorithm returns Fail with probability at most $O(n_0 \log n_0) \cdot n_0^{-10} \leq O(n_0^{-8})$. □

Proof of Theorem 7.5. To compute the claimed strong LDD we call StrongLDD$(G, \frac{1}{2}D)$ with the following two modifications:

First, whenever some recursive call returns Fail, we simply restart the whole algorithm.

Second, we test whether the returned set of edges $S \subseteq E(G)$ satisfies the Strong Diameter property. To this end, we compute the strongly connected components in $G \setminus S$ and compute, for any such component C, a 2-approximation of its diameter. By a standard argument, such a 2-approximation can be obtained in Dijkstra-time by (1) selecting an arbitrary node v, (2) computing $d^{out} := \max_{u \in V(G)} d_G(v,u)$ by solving SSSP on G, (3) computing $d^{in} := \max_{u \in V(G)} d_G(u,v)$ by solving SSSP on the reversed graph of G, and returning $\max\{d^{in}, d^{out}\}$. If the diameter approximations are at most $\frac{D}{2}$ in all components, we return S. Otherwise, we restart the whole algorithm.

This algorithm indeed never fails to satisfy the Strong Diameter property: Since the diameter approximations have approximation factor at most 2, we have certified that the diameter of any strongly connected component is at most D in the graph $G \setminus S$. Moreover, with high probability the execution of Algorithm 22 passes both tests with high probability (by Lemmas 9.8 and 9.11), and therefore we expect to repeat the algorithm $O(1)$ times. Since the repetitions are independent of each other, the edge hitting probability increases only by a constant factor and remains $O(\frac{w(e)}{D} \cdot \log^3(n_0))$ by Lemma 9.9.

Finally, consider the running time. As argued before, with high probability we avoid restarting Algorithm 22 altogether. Thus, with high probability the algorithm runs in total time is $O((m + n_0 \log\log n_0) \log^2(n_0))$ by Lemma 9.10. Since we expect to repeat the algorithm at most $O(1)$ times, the same bound applies to the expected running time. □

Bibliography

[Abb+16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. “Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made.” In: *STOC*. ACM, 2016, pp. 375–388. DOI: 10.1145/2897518.2897653.

[Abb+22a] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. “Scheduling lower bounds via AND subset sum.” In: *J. Comput. Syst. Sci.* 127 (2022), pp. 29–40. DOI: 10.1016/J.JCSS.2022.01.005.

[Abb+22b] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. “SETH-based Lower Bounds for Subset Sum and Bicriteria Path.” In: *ACM Trans. Algorithms* 18.1 (2022), 6:1–6:22. DOI: 10.1145/3450524.

[ABW15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. “Tight Hardness Results for LCS and Other Sequence Similarity Measures.” In: *FOCS*. IEEE Computer Society, 2015, pp. 59–78. DOI: 10.1109/FOCS.2015.14.

[Agg+87] Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. “Geometric Applications of a Matrix-Searching Algorithm.” In: *Algorithmica* 2 (1987), pp. 195–208. DOI: 10.1007/BF01840359.

[Ahu+90] Ravindra K. Ahuja, Kurt Mehlhorn, James B. Orlin, and Robert Endre Tarjan. “Faster Algorithms for the Shortest Path Problem.” In: *J. ACM* 37.2 (1990), pp. 213–223. DOI: 10.1145/77600.77615.

[AKO10] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. “Polylogarithmic Approximation for Edit Distance and the Asymmetric Query Complexity.” In: *FOCS*. IEEE Computer Society, 2010, pp. 377–386. DOI: 10.1109/FOCS.2010.43.

[AKO11] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. “Streaming Algorithms via Precision Sampling.” In: *FOCS*. IEEE Computer Society, 2011, pp. 363–372. DOI: 10.1109/FOCS.2011.82.

[Alo+92] Noga Alon, Zvi Galil, Oded Margalit, and Moni Naor. “Witnesses for Boolean Matrix Multiplication and for Shortest Paths.” In: *FOCS*. IEEE Computer Society, 1992, pp. 417–426. DOI: 10.1109/SFCS.1992.267748.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows – Theory, algorithms and applications*. Prentice Hall, 1993. ISBN: 978-0-13-617549-0.

[AMV20] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. “Circulation Control for Faster Minimum Cost Flow in Unit-Capacity Graphs.” In: *FOCS*. IEEE, 2020, pp. 93–104. DOI: 10.1109/FOCS46700.2020.00018.

[AN10] Alexandr Andoni and Huy L. Nguyen. “Near-Optimal Sublinear Time Algorithms for Ulam Distance.” In: *SODA*. SIAM, 2010, pp. 76–86. DOI: 10.1137/1.9781611973075.8.

[AN20] Alexandr Andoni and Negev Shekel Nosatzki. “Edit Distance in Near-Linear Time: it’s a Constant Factor.” In: *FOCS*. IEEE, 2020, pp. 990–1001. DOI: 10.1109/FOCS46700.2020.00096.

[AN96] Noga Alon and Moni Naor. “Derandomization, Witnesses for Boolean Matrix Multiplication and Construction of Perfect Hash Functions.” In: *Algorithmica* 16.4/5 (1996), pp. 434–449. DOI: 10.1007/BF01940874.

[And17] Alexandr Andoni. “High frequency moments via max-stability.” In: *ICASSP*. IEEE, 2017, pp. 6364–6368. DOI: 10.1109/ICASSP.2017.7953381.

[And20] Alex Andoni. “Simpler Constant-Factor Approximation to Edit Distance.” 2020. URL: https://www.cs.columbia.edu/~andoni/papers/edit/simple.pdf (visited on 02/19/2024).

[AO12] Alexandr Andoni and Krzysztof Onak. “Approximating Edit Distance in Near-Linear Time.” In: *SIAM J. Comput.* 41.6 (2012), pp. 1635–1648. DOI: 10.1137/090767182.

[AP92] Baruch Awerbuch and David Peleg. “Routing with Polynomial Communication-Space Trade-Off.” In: *SIAM J. Discret. Math.* 5.2 (1992), pp. 151–162. DOI: 10.1137/0405013.

[AT19] Kyriakos Axiotis and Christos Tzamos. “Capacitated Dynamic Programming: Faster Knapsack and Graph Algorithms.” In: *ICALP*. Vol. 132. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 19:1–19:13. DOI: 10.4230/LIPIcs.ICALP.2019.19.

[Awe+89] Baruch Awerbuch, Andrew V. Goldberg, Michael Luby, and Serge A. Plotkin. “Network Decomposition and Locality in Distributed Computation.” In: *FOCS*. IEEE Computer Society, 1989, pp. 364–369. DOI: 10.1109/SFCS.1989.63504.

[Awe+92] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. “Fast Network Decomposition (Extended Abstract).” In: *PODC*. ACM, 1992, pp. 169–177. DOI: 10.1145/135419.135456.

[Awe85] Baruch Awerbuch. “Complexity of Network Synchronization.” In: *J. ACM* 32.4 (1985), pp. 804–823. DOI: 10.1145/4221.4227.

[Axi+21] Kyriakos Axiotis, Arturs Backurs, Karl Bringmann, Ce Jin, Vasileios Nakos, Christos Tzamos, and Hongxun Wu. “Fast and Simple Modular Subset Sum.” In: *SOSA*. SIAM, 2021, pp. 57–67. DOI: 10.1137/1.9781611976496.6.

[Bar+04] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. "Approximating Edit Distance Efficiently." In: *FOCS*. IEEE Computer Society, 2004, pp. 550–559. DOI: 10.1109/FOCS.2004.14.

[Bar96] Yair Bartal. "Probabilistic Approximations of Metric Spaces and Its Algorithmic Applications." In: *FOCS*. IEEE Computer Society, 1996, pp. 184–193. DOI: 10.1109/SFCS.1996.548477.

[Bat+03] Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. "A sublinear algorithm for weakly approximating edit distance." In: *STOC*. ACM, 2003, pp. 316–324. DOI: 10.1145/780542.780590.

[Bat+18] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. "Fast algorithms for knapsack via convolution and prediction." In: *STOC*. ACM, 2018, pp. 1269–1282. DOI: 10.1145/3188745.3188876.

[BC22] Karl Bringmann and Alejandro Cassis. "Faster Knapsack Algorithms via Bounded Monotone Min-Plus-Convolution." In: *ICALP*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 31:1–31:21. DOI: 10.4230/LIPIcs.ICALP.2022.31.

[BC23] Karl Bringmann and Alejandro Cassis. "Faster 0-1-Knapsack via Near-Convex Min-Plus-Convolution." In: *ESA*. Vol. 274. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 24:1–24:16. DOI: 10.4230/LIPIcs.ESA.2023.24.

[BC92] Peter Butkovic and Raymond A. Cuninghame-Green. "An $O(n^2)$ algorithm for the maximum cycle mean of an $n \times n$ bivalent matrix." In: *Discret. Appl. Math.* 35.2 (1992), pp. 157–162. DOI: 10.1016/0166-218X(92)90039-D.

[BCF23] Karl Bringmann, Alejandro Cassis, and Nick Fischer. "Negative-Weight Single-Source Shortest Paths in Near-Linear Time: Now Faster!" In: *FOCS*. IEEE, 2023, pp. 515–538. DOI: 10.1109/FOCS57990.2023.00038.

[BCR20] Joshua Brakensiek, Moses Charikar, and Aviad Rubinstein. "A Simple Sublinear Algorithm for Gap Edit Distance." In: *CoRR* abs/2007.14368 (2020). DOI: 10.48550/arXiv.2007.14368.

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1957. DOI: 10.2307/j.ctv1nxcw0f.

[Bel58] Richard Bellman. "On a Routing Problem." In: *Quarterly of Applied Mathematics* 16.1 (1958), pp. 87–90. DOI: 10.1090/qam/102435.

[BES06] Tugkan Batu, Funda Ergün, and Süleyman Cenk Sahinalp. "Oblivious string embeddings and edit distance approximations." In: *SODA*. ACM Press, 2006, pp. 792–801. DOI: 10.1145/1109557.1109644.

[BF13] Karl Bringmann and Tobias Friedrich. "Exact and Efficient Generation of Geometric Random Variates and Random Graphs." In: *ICALP (1)*. Vol. 7965. Lecture Notes in Computer Science. Springer, 2013, pp. 267–278. DOI: 10.1007/978-3-642-39206-1_23.

[BFN21] Karl Bringmann, Nick Fischer, and Vasileios Nakos. "Sparse nonnegative convolution is equivalent to dense nonnegative convolution." In: *STOC*. ACM, 2021, pp. 1711–1724. DOI: 10.1145/3406325.3451090.

[BFN22] Karl Bringmann, Nick Fischer, and Vasileios Nakos. "Deterministic and Las Vegas Algorithms for Sparse Nonnegative Convolution." In: *SODA*. SIAM, 2022, pp. 3069–3090. DOI: 10.1137/1.9781611977073.119.

[BGW20] Aaron Bernstein, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. "Near-Optimal Decremental SSSP in Dense Weighted Digraphs." In: *FOCS*. IEEE, 2020, pp. 1112–1122. DOI: 10.1109/FOCS46700.2020.00107.

[BI18] Arturs Backurs and Piotr Indyk. "Edit Distance Cannot Be Computed in Strongly Subquadratic Time (Unless SETH is False)." In: *SIAM J. Comput.* 47.3 (2018), pp. 1087–1097. DOI: 10.1137/15M1053128.

[BIS17] Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. "Better Approximations for Tree Sparsity in Nearly-Linear Time." In: *SODA*. SIAM, 2017, pp. 2215–2229. DOI: 10.1137/1.9781611974782.145.

[BK15] Karl Bringmann and Marvin Künnemann. "Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping." In: *FOCS*. IEEE Computer Society, 2015, pp. 79–97. DOI: 10.1109/FOCS.2015.15.

[Ble+14] Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, Gary L. Miller, Richard Peng, and Kanat Tangwongsan. "Nearly-Linear Work Parallel SDD Solvers, Low-Diameter Decomposition, and Low-Stretch Subgraphs." In: *Theory Comput. Syst.* 55.3 (2014), pp. 521–554. DOI: 10.1007/s00224-013-9444-5.

[BN20] Karl Bringmann and Vasileios Nakos. "Top-k-convolution and the quest for near-linear output-sensitive subset sum." In: *STOC*. ACM, 2020, pp. 982–995. DOI: 10.1145/3357713.3384308.

[BN21] Karl Bringmann and Vasileios Nakos. "A Fine-Grained Perspective on Approximating Subset Sum and Partition." In: *SODA*. SIAM, 2021, pp. 1797–1815. DOI: 10.1137/1.9781611976465.108.

[BNW22] Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. "Negative-Weight Single-Source Shortest Paths in Near-linear Time." In: *FOCS*. IEEE, 2022, pp. 600–611. DOI: 10.1109/FOCS54457.2022.00063.

[Bor+21] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. "Approximating Edit Distance in Truly Subquadratic Time: Quantum and MapReduce." In: *J. ACM* 68.3 (2021), 19:1–19:41. DOI: 10.1145/3456807.

[BR20] Joshua Brakensiek and Aviad Rubinstein. "Constant-factor approximation of near-linear edit distance in near-linear time." In: *STOC*. ACM, 2020, pp. 685–698. DOI: 10.1145/3357713.3384282.

[Bra+20] Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. "Bipartite Matching in Nearly-linear Time on Moderately Dense Graphs." In: *FOCS*. IEEE, 2020, pp. 919–930. DOI: 10.1109/FOCS46700.2020.00090.

[Bra+21] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. "Minimum cost flows, MDPs, and ℓ_1-regression in nearly linear time for dense instances." In: *STOC*. ACM, 2021, pp. 859–869. DOI: 10.1145/3406325.3451108.

[Bra+23] Jan van den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P. Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. "A Deterministic Almost-Linear Time Algorithm for Minimum-Cost Flow." In: *FOCS*. IEEE, 2023, pp. 503–514. DOI: 10.1109/FOCS57990.2023.00037.

[Bre+14] David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. "Necklaces, Convolutions, and X+Y." In: *Algorithmica* 69.2 (2014), pp. 294–314. DOI: 10.1007/s00453-012-9734-3.

[Bri+21] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Marvin Künnemann. "Fine-Grained Completeness for Optimization in P." In: *APPROX-RANDOM*. Vol. 207. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 9:1–9:22. DOI: 10.4230/LIPIcs.APPROX/RANDOM.2021.9.

[Bri+22a] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Marvin Künnemann. "A Structural Investigation of the Approximability of Polynomial-Time Problems." In: *ICALP*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 30:1–30:20. DOI: 10.4230/LIPIcs.ICALP.2022.30.

[Bri+22b] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. "Almost-optimal sublinear-time edit distance in the low distance regime." In: *STOC*. ACM, 2022, pp. 1102–1115. DOI: 10.1145/3519935.3519990.

[Bri+22c] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Vasileios Nakos. "Improved Sublinear-Time Edit Distance for Preprocessed Strings." In: *ICALP*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 32:1–32:20. DOI: 10.4230/LIPIcs.ICALP.2022.32.

[Bri+22d] Karl Bringmann, Nick Fischer, Danny Hermelin, Dvir Shabtay, and Philip Wellnitz. "Faster Minimization of Tardy Processing Time on a Single Machine." In: *Algorithmica* 84.5 (2022), pp. 1341–1356. DOI: `10.1007/S00453-022-00928-W`.

[Bri+24] Karl Bringmann, Alejandro Cassis, Nick Fischer, and Tomasz Kociumaka. "Faster Sublinear-Time Edit Distance." In: *SODA*. SIAM, 2024, pp. 3274–3301. DOI: `10.1137/1.9781611977912.117`.

[Bri17] Karl Bringmann. "A Near-Linear Pseudopolynomial Time Algorithm for Subset Sum." In: *SODA*. SIAM, 2017, pp. 1073–1084. DOI: `10.1137/1.9781611974782.69`.

[Bri23] Karl Bringmann. "Knapsack with Small Items in Near-Quadratic Time." In: *CoRR* abs/2308.03075 (2023). DOI: `10.48550/arXiv.2308.03075`.

[Bus+94] Michael R. Bussieck, Hannes Hassler, Gerhard J. Woeginger, and Uwe T. Zimmermann. "Fast algorithms for the maximum convolution problem." In: *Oper. Res. Lett.* 15.3 (1994), pp. 133–141. DOI: `10.1016/0167-6377(94)90048-5`.

[BW21] Karl Bringmann and Philip Wellnitz. "On Near-Linear-Time Algorithms for Dense Subset Sum." In: *SODA*. SIAM, 2021, pp. 1777–1796. DOI: `10.1137/1.9781611976465.107`.

[CG99] Boris V. Cherkassky and Andrew V. Goldberg. "Negative-cycle detection algorithms." In: *Math. Program.* 85.2 (1999), pp. 277–311. DOI: `10.1007/s101070050058`.

[CGS99] Boris V. Cherkassky, Andrew V. Goldberg, and Craig Silverstein. "Buckets, Heaps, Lists, and Monotone Priority Queues." In: *SIAM J. Comput.* 28.4 (1999), pp. 1326–1346. DOI: `10.1137/S0097539796313490`.

[CH02] Richard Cole and Ramesh Hariharan. "Verifying candidate matches in sparse and wildcard matching." In: *STOC*. ACM, 2002, pp. 592–601. DOI: `10.1145/509907.509992`.

[CH21] Timothy M. Chan and Sariel Har-Peled. "Smallest k-Enclosing Rectangle Revisited." In: *Discret. Comput. Geom.* 66.2 (2021), pp. 769–791. DOI: `10.1007/s00454-020-00239-3`.

[CH22] Timothy M. Chan and Qizheng He. "More on change-making and related problems." In: *J. Comput. Syst. Sci.* 124 (2022), pp. 159–169. DOI: `10.1016/j.jcss.2021.09.005`.

[Cha+14] Krishnendu Chatterjee, Monika Henzinger, Sebastian Krinninger, Veronika Loitzenbauer, and Michael A. Raskin. "Approximating the minimum cycle mean." In: *Theor. Comput. Sci.* 547 (2014), pp. 104–116. DOI: `10.1016/j.tcs.2014.06.031`.

[Cha+20] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. "Approximating Edit Distance Within Constant Factor in Truly Sub-quadratic Time." In: *J. ACM* 67.6 (2020), 36:1–36:22. DOI: 10.1145/3422823.

[Cha18] Timothy M. Chan. "Approximation Schemes for 0-1 Knapsack." In: *SOSA*. Vol. 61. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 5:1–5:12. DOI: 10.4230/OASIcs.SOSA.2018.5.

[Che+08] Boris V. Cherkassky, Loukas Georgiadis, Andrew V. Goldberg, Robert Endre Tarjan, and Renato Fonseca F. Werneck. "Shortest Path Feasibility Algorithms: An Experimental Evaluation." In: *ALENEX*. SIAM, 2008, pp. 118–132. DOI: 10.1137/1.9781611972887.12.

[Che+22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time." In: *FOCS*. IEEE, 2022, pp. 612–623. DOI: 10.1109/FOCS54457.2022.00064.

[Che+23] Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. "A Nearly Quadratic-Time FPTAS for Knapsack." In: *CoRR* abs/2308.07821 (2023). DOI: 10.48550/arXiv.2308.07821.

[Che+24] Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. "Faster Algorithms for Bounded Knapsack and Bounded Subset Sum Via Fine-Grained Proximity Results." In: *SODA*. SIAM, 2024, pp. 4828–4848. DOI: 10.1137/1.9781611977912.171.

[Chi+22] Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. "Faster min-plus product for monotone instances." In: *STOC*. ACM, 2022, pp. 1529–1542. DOI: 10.1145/3519935.3520057.

[CI21] Jean Cardinal and John Iacono. "Modular Subset Sum, Dynamic Strings, and Zero-Sum Sets." In: *SOSA*. SIAM, 2021, pp. 45–56. DOI: 10.1137/1.9781611976496.5.

[CK24] Julia Chuzhoy and Sanjeev Khanna. "A Faster Combinatorial Algorithm for Maximum Bipartite Matching." In: *SODA*. SIAM, 2024, pp. 2185–2235. DOI: 10.1137/1.9781611977912.79.

[CKW23] Alejandro Cassis, Tomasz Kociumaka, and Philip Wellnitz. "Optimal Algorithms for Bounded Weighted Edit Distance." In: *FOCS*. IEEE, 2023, pp. 2177–2187. DOI: 10.1109/FOCS57990.2023.00135.

[CL15] Timothy M. Chan and Moshe Lewenstein. "Clustered Integer 3SUM via Additive Combinatorics." In: *STOC*. ACM, 2015, pp. 31–40. DOI: 10.1145/2746539.2746568.

[Coh+17] Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. "Negative-Weight Shortest Paths and Unit Capacity Minimum Cost Flow in $\widetilde{O}(m^{10/7} \log W)$ Time (Extended Abstract)." In: *SODA*. SIAM, 2017, pp. 752–771. DOI: 10.1137/1.9781611974782.48.

[Con23] Wikipedia Contributors. *Shortest Path Faster Algorithm*. 2023. URL: https://en.wikipedia.org/wiki/Shortest_path_faster_algorithm (visited on 04/01/2023).

[CW21] Timothy M. Chan and R. Ryan Williams. "Deterministic APSP, Orthogonal Vectors, and More: Quickly Derandomizing Razborov-Smolensky." In: *ACM Trans. Algorithms* 17.1 (2021), 2:1–2:14. DOI: 10.1145/3402926.

[Cyg+16] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. "On Problems as Hard as CNF-SAT." In: *ACM Trans. Algorithms* 12.3 (2016), 41:1–41:24. DOI: 10.1145/2925416.

[Cyg+19] Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. "On Problems Equivalent to (min, +)-Convolution." In: *ACM Trans. Algorithms* 15.1 (2019), 14:1–14:25. DOI: 10.1145/3293465.

[CZ20] Shiri Chechik and Tianyi Zhang. "Dynamic Low-Stretch Spanning Trees in Subpolynomial Time." In: *SODA*. SIAM, 2020, pp. 463–475. DOI: 10.1137/1.9781611975994.28.

[Dij59] Edsger W. Dijkstra. "A note on two problems in connexion with graphs." In: *Numerische Mathematik* 1 (1959), pp. 269–271. DOI: 10.1007/BF01386390.

[DJM23] Mingyang Deng, Ce Jin, and Xiao Mao. "Approximating Knapsack and Partition via Dense Subset Sums." In: *SODA*. SIAM, 2023, pp. 2961–2979. DOI: 10.1137/1.9781611977554.ch113.

[DMZ23] Mingyang Deng, Xiao Mao, and Ziqian Zhong. "On Problems Related to Unbounded SubsetSum: A Unified Combinatorial Approach." In: *SODA*. SIAM, 2023, pp. 2980–2990. DOI: 10.1137/1.9781611977554.ch114.

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. DOI: 10.1017/CBO9780511581274.

[Edw23] Chris Edwards. "Historic Algorithms Help Unlock Shortest-Path Problem Breakthrough." In: *Commun. ACM* 66.9 (2023), pp. 10–12. DOI: 10.1145/3607866.

[EKZ77] Peter van Emde Boas, R. Kaas, and E. Zijlstra. "Design and Implementation of an Efficient Priority Queue." In: *Math. Syst. Theory* 10 (1977), pp. 99–127. DOI: 10.1007/BF01683268.

[Emd77] Peter van Emde Boas. “Preserving Order in a Forest in Less Than Logarithmic Time and Linear Space.” In: *Inf. Process. Lett.* 6.3 (1977), pp. 80–82. DOI: 10.1016/0020-0190(77)90031-X.

[EW20] Friedrich Eisenbrand and Robert Weismantel. “Proximity Results and Faster Algorithms for Integer Programming Using the Steinitz Lemma.” In: *ACM Trans. Algorithms* 16.1 (2020), 5:1–5:14. DOI: 10.1145/3340322.

[Fei+94] Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. “Computing with Noisy Information.” In: *SIAM J. Comput.* 23.5 (1994), pp. 1001–1018. DOI: 10.1137/S0097539791195877.

[FG19] Sebastian Forster and Gramoz Goranci. “Dynamic low-stretch trees via dynamic low-diameter decompositions.” In: *STOC*. ACM, 2019, pp. 377–388. DOI: 10.1145/3313276.3316381.

[FGV21] Sebastian Forster, Martin Grösbacher, and Tijn de Vos. “An Improved Random Shift Algorithm for Spanners and Low Diameter Decompositions.” In: *OPODIS*. Vol. 217. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 16:1–16:17. DOI: 10.4230/LIPIcs.OPODIS.2021.16.

[Fin23] Jeremy T. Fineman. “Single-Source Shortest Paths with Negative Real Weights in $\tilde{O}(mn^{8/9})$-Time.” In: *CoRR* abs/2311.02520 (2023). DOI: 10.48550/arXiv.2311.02520.

[For56] Lester Ford. “Network Flow Theory.” In: *Paper P-923, California, RAND Corporation* (1956). URL: https://www.rand.org/pubs/papers/P923.html.

[FR06] Jittat Fakcharoenphol and Satish Rao. “Planar graphs, negative weight edges, shortest paths, and near linear time.” In: *J. Comput. Syst. Sci.* 72.5 (2006), pp. 868–889. DOI: 10.1016/j.jcss.2005.05.007.

[FT87] Michael L. Fredman and Robert Endre Tarjan. “Fibonacci heaps and their uses in improved network optimization algorithms.” In: *J. ACM* 34.3 (1987), pp. 596–615. DOI: 10.1145/28869.28874.

[FW65] N. J. Fine and H. S. Wilf. “Uniqueness Theorems for Periodic Functions.” In: *Proceedings of the American Mathematical Society* 16.1 (1965), pp. 109–114. ISSN: 00029939, 10886826. URL: http://www.jstor.org/stable/2034009 (visited on 07/13/2023).

[FW93] Michael L. Fredman and Dan E. Willard. “Surpassing the Information Theoretic Bound with Fusion Trees.” In: *J. Comput. Syst. Sci.* 47.3 (1993), pp. 424–436. DOI: 10.1016/0022-0000(93)90040-4.

[FW94] Michael L. Fredman and Dan E. Willard. “Trans-Dichotomous Algorithms for Minimum Spanning Trees and Shortest Paths.” In: *J. Comput. Syst. Sci.* 48.3 (1994), pp. 533–551. DOI: 10.1016/S0022-0000(05)80064-9.

[Gab83] Harold N. Gabow. “Scaling Algorithms for Network Problems.” In: *FOCS*. IEEE Computer Society, 1983, pp. 248–257. DOI: 10.1109/SFCS.1983.68.

[GGC20] Pascal Giorgi, Bruno Grenet, and Armelle Perret du Cray. “Essentially optimal sparse polynomial multiplication.” In: *ISSAC*. ACM, 2020, pp. 202–209. DOI: 10.1145/3373207.3404026.

[GKS19] Elazar Goldenberg, Robert Krauthgamer, and Barna Saha. “Sublinear Algorithms for Gap Edit Distance.” In: *FOCS*. IEEE Computer Society, 2019, pp. 1101–1120. DOI: 10.1109/FOCS.2019.00070.

[GM91] Zvi Galil and Oded Margalit. “An Almost Linear-Time Algorithm for the Dense Subset-Sum Problem.” In: *SIAM J. Comput.* 20.6 (1991), pp. 1157–1189. DOI: 10.1137/0220072.

[Gol+22] Elazar Goldenberg, Tomasz Kociumaka, Robert Krauthgamer, and Barna Saha. “Gap Edit Distance via Non-Adaptive Queries: Simple and Optimal.” In: *FOCS*. IEEE, 2022, pp. 674–685. DOI: 10.1109/FOCS54457.2022.00070.

[Gol95] Andrew V. Goldberg. “Scaling Algorithms for the Shortest Paths Problem.” In: *SIAM J. Comput.* 24.3 (1995), pp. 494–504. DOI: 10.1137/S0097539792231179.

[Gra16] Szymon Grabowski. “New tabulation and sparse dynamic programming based techniques for sequence similarity problems.” In: *Discret. Appl. Math.* 212 (2016), pp. 96–103. DOI: 10.1016/j.dam.2015.10.040.

[Gra72] Ronald L. Graham. “An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set.” In: *Inf. Process. Lett.* 1.4 (1972), pp. 132–133. DOI: 10.1016/0020-0190(72)90045-2.

[GRS20] Elazar Goldenberg, Aviad Rubinstein, and Barna Saha. “Does preprocessing help in fast sequence comparisons?” In: *STOC*. ACM, 2020, pp. 657–670. DOI: 10.1145/3357713.3384300.

[GS80] V. S. Grinberg and S. V. Sevastyanov. “Value of the Steinitz constant.” In: *Funktsional. Anal. i Prilozhen.* 14.2 (1980), pp. 56–57. ISSN: 0374-1990. DOI: 10.1007/BF01086559.

[GT89] Harold N. Gabow and Robert Endre Tarjan. “Faster Scaling Algorithms for Network Problems.” In: *SIAM J. Comput.* 18.5 (1989), pp. 1013–1036. DOI: 10.1137/0218069.

[Hen+97] Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. “Faster Shortest-Path Algorithms for Planar Graphs.” In: *J. Comput. Syst. Sci.* 55.1 (1997), pp. 3–23. DOI: 10.1006/jcss.1997.1493.

[HMS22] Danny Hermelin, Hendrik Molter, and Dvir Shabtay. “Minimizing the Weighted Number of Tardy Jobs via (max,+)-Convolutions.” In: *CoRR* abs/2202.06841 (2022). DOI: 10.48550/arXiv.2202.06841.

[HX24] Qizheng He and Zhean Xu. “Simple and Faster Algorithms for Knapsack.” In: *SOSA*. SIAM, 2024, pp. 56–62. DOI: 10.1137/1.9781611977936.6.

[IK75] Oscar H. Ibarra and Chul E. Kim. “Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems.” In: *J. ACM* 22.4 (1975), pp. 463–468. DOI: 10.1145/321906.321909.

[IP01] Russell Impagliazzo and Ramamohan Paturi. “On the Complexity of k-SAT.” In: *J. Comput. Syst. Sci.* 62.2 (2001), pp. 367–375. DOI: 10.1006/jcss.2000.1727.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. “Which Problems Have Strongly Exponential Complexity?” In: *J. Comput. Syst. Sci.* 63.4 (2001), pp. 512–530. DOI: 10.1006/jcss.2001.1774.

[IW05] Piotr Indyk and David P. Woodruff. “Optimal approximations of the frequency moments of data streams.” In: *STOC*. ACM, 2005, pp. 202–208. DOI: 10.1145/1060590.1060621.

[Jin19] Ce Jin. “An Improved FPTAS for 0-1 Knapsack.” In: *ICALP*. Vol. 132. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 76:1–76:14. DOI: 10.4230/LIPICS.ICALP.2019.76.

[Jin23a] Ce Jin. “0-1 Knapsack in Nearly Quadratic Time.” In: *CoRR* abs/2308.04093 (2023). DOI: 10.48550/arXiv.2308.04093.

[Jin23b] Ce Jin. “Solving Knapsack with Small Items via L0-Proximity.” In: *CoRR* abs/2307.09454 (2023). DOI: 10.48550/arXiv.2307.09454.

[JK18] Klaus Jansen and Stefan Erich Julius Kraft. “A faster FPTAS for the Unbounded Knapsack Problem.” In: *Eur. J. Comb.* 68 (2018), pp. 148–174. DOI: 10.1016/j.ejc.2017.07.016.

[Joh77] Donald B. Johnson. “Efficient Algorithms for Shortest Paths in Sparse Networks.” In: *J. ACM* 24.1 (1977), pp. 1–13. DOI: 10.1145/321992.321993.

[JR23] Klaus Jansen and Lars Rohwedder. “On Integer Programming, Discrepancy, and Convolution.” In: *Math. Oper. Res.* 48.3 (2023), pp. 1481–1495. DOI: 10.1287/moor.2022.1308.

[JW19] Ce Jin and Hongxun Wu. “A Simple Near-Linear Pseudopolynomial Time Randomized Algorithm for Subset Sum.” In: *SOSA*. Vol. 69. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 17:1–17:6. DOI: 10.4230/OASIcs.SOSA.2019.17.

[Kar72] Richard M. Karp. “Reducibility Among Combinatorial Problems.” In: *Complexity of Computer Computations.* The IBM Research Symposia Series. Plenum Press, New York, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[Kar78] Richard M. Karp. “A characterization of the minimum cycle mean in a digraph.” In: *Discret. Math.* 23.3 (1978), pp. 309–311. DOI: 10.1016/0012-365X(78)90011-0.

[KJP77] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. “Fast Pattern Matching in Strings.” In: *SIAM J. Comput.* 6.2 (1977), pp. 323–350. DOI: 10.1137/0206024.

[Kle22] Kim-Manuel Klein. “On the Fine-Grained Complexity of the Unbounded SubsetSum and the Frobenius Problem.” In: *SODA*. SIAM, 2022, pp. 3567–3582. DOI: 10.1137/1.9781611977073.141.

[KMS23] Tomasz Kociumaka, Anish Mukherjee, and Barna Saha. “Approximating Edit Distance in the Fully Dynamic Model.” In: *FOCS*. IEEE, 2023, pp. 1628–1638. DOI: 10.1109/FOCS57990.2023.00098.

[KMW09] Philip N. Klein, Shay Mozes, and Oren Weimann. “Shortest paths in directed planar graphs with negative lengths: a linear-space $O(n \log^2 n)$-time algorithm.” In: *SODA*. SIAM, 2009, pp. 236–245. DOI: 10.1137/1.9781611973068.27.

[KNS24] Adam Karczmarz, Wojciech Nadara, and Marek Sokolowski. “Exact Shortest Paths with Rational Weights on the Word RAM.” In: *SODA*. SIAM, 2024, pp. 2597–2608. DOI: 10.1137/1.9781611977912.92.

[KP04] Hans Kellerer and Ulrich Pferschy. “Improved Dynamic Programming in Connection with an FPTAS for the Knapsack Problem.” In: *J. Comb. Optim.* 8.1 (2004), pp. 5–11. DOI: 10.1023/B:JOCO.0000021934.29833.6b.

[KP23] Tomasz Kociumaka and Adam Polak. “Bellman-Ford Is Optimal for Shortest Hop-Bounded Paths.” In: *ESA*. Vol. 274. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 72:1–72:10. DOI: 10.4230/LIPIcs.ESA.2023.72.

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

[KPR23] Kim-Manuel Klein, Adam Polak, and Lars Rohwedder. “On Minimizing Tardy Processing Time, Max-Min Skewed Convolution, and Triangular Structured ILPs.” In: *SODA*. SIAM, 2023, pp. 2947–2960. DOI: 10.1137/1.9781611977554.CH112.

[KPS17] Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. “On the Fine-Grained Complexity of One-Dimensional Dynamic Programming.” In: *ICALP*. Vol. 80. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 21:1–21:15. DOI: 10.4230/LIPIcs.ICALP.2017.21.

[KS20a] Tomasz Kociumaka and Barna Saha. “Sublinear-Time Algorithms for Computing & Embedding Gap Edit Distance.” In: *FOCS*. IEEE, 2020, pp. 1168–1179. DOI: 10.1109/FOCS46700.2020.00112.

[KS20b] Michal Koucký and Michael E. Saks. "Constant factor approximations to edit distance on far input pairs in nearly linear time." In: *STOC*. ACM, 2020, pp. 699–712. DOI: 10.1145/3357713.3384307.

[KX19] Konstantinos Koiliaris and Chao Xu. "Faster Pseudopolynomial Time Algorithms for Subset Sum." In: *ACM Trans. Algorithms* 15.3 (2019), 40:1–40:20. DOI: 10.1145/3329863.

[Law66] Eugene L Lawler. "Optimal cycles in doubly weighted directed linear graphs." In: *Proc. Int'l Symp. Theory of Graphs*. 1966, pp. 209–232.

[Law76] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976. ISBN: 9780030848667.

[Law79] Eugene L. Lawler. "Fast Approximation Algorithms for Knapsack Problems." In: *Math. Oper. Res.* 4.4 (1979), pp. 339–356. DOI: 10.1287/moor.4.4.339.

[Len17] Johannes Lengler. "Drift Analysis." In: *CoRR* abs/1712.00964 (2017). DOI: 10.48550/arXiv.1712.00964.

[Lev66] Vladimir Iosifovich Levenshtein. "Binary codes capable of correcting deletions, insertions and reversals." In: *Soviet Physics Doklady* 10.8 (1966), pp. 707–710.

[LMS98] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. "Incremental String Comparison." In: *SIAM J. Comput.* 27.2 (1998), pp. 557–582. DOI: 10.1137/S0097539794264810.

[LRC14] Eduardo Sany Laber, Wilfredo Bardales Roncalla, and Ferdinando Cicalese. "On lower bounds for the Maximum Consecutive Subsums Problem and the (min, +)-convolution." In: *ISIT*. IEEE, 2014, pp. 1807–1811. DOI: 10.1109/ISIT.2014.6875145.

[LRT79] Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. "Generalized Nested Dissection." In: *SIAM Journal on Numerical Analysis* 16.2 (1979), pp. 346–358. DOI: 10.1137/0716027.

[LS93] Nathan Linial and Michael E. Saks. "Low diameter graph decompositions." In: *Comb.* 13.4 (1993), pp. 441–454. DOI: 10.1007/BF01303516.

[LV88] Gad M. Landau and Uzi Vishkin. "Fast String Matching with k Differences." In: *J. Comput. Syst. Sci.* 37.1 (1988), pp. 63–78. DOI: 10.1016/0022-0000(88)90045-1.

[Mao23] Xiao Mao. "(1-ϵ)-Approximation of Knapsack in Nearly Quadratic Time." In: *CoRR* abs/2308.07004 (2023). DOI: 10.48550/arXiv.2308.07004.

[Mat10] Jiří Matoušek. *Thirty-three miniatures: Mathematical and Algorithmic applications of Linear Algebra*. American Mathematical Society Providence, RI, 2010. DOI: 10.1090/stml/053.

[MN90] Kurt Mehlhorn and Stefan Näher. "Bounded Ordered Dictionaries in $O(\log\log N)$ Time and $O(n)$ Space." In: *Inf. Process. Lett.* 35.4 (1990), pp. 183–189. DOI: 10.1016/0020-0190(90)90022-P.

[Moo59] Edward F. Moore. "The Shortest Path Through a Maze." In: *Proceedings of the International Symposium on the Theory of Switching*. Cambridge, Harvard University Press, 1959, pp. 285–292.

[MP80] William J. Masek and Mike Paterson. "A Faster Algorithm Computing String Edit Distances." In: *J. Comput. Syst. Sci.* 20.1 (1980), pp. 18–31. DOI: 10.1016/0022-0000(80)90002-1.

[MPX13] Gary L. Miller, Richard Peng, and Shen Chen Xu. "Parallel graph decompositions using random shifts." In: *SPAA*. ACM, 2013, pp. 196–203. DOI: 10.1145/2486159.2486180.

[MWW19] Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. "A Subquadratic Approximation Scheme for Partition." In: *SODA*. SIAM, 2019, pp. 70–88. DOI: 10.1137/1.9781611975482.5.

[Mye86] Eugene W. Myers. "An O(ND) Difference Algorithm and Its Variations." In: *Algorithmica* 1.2 (1986), pp. 251–266. DOI: 10.1007/BF01840446.

[Nak20] Vasileios Nakos. "Nearly Optimal Sparse Polynomial Multiplication." In: *IEEE Trans. Inf. Theory* 66.11 (2020), pp. 7231–7236. DOI: 10.1109/TIT.2020.2989385.

[NW70] Saul B. Needleman and Christian D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins." In: *Journal of molecular biology* 48.3 (1970), pp. 443–453. DOI: 10.1016/b978-0-12-131200-8.50031-9.

[OA92] James B. Orlin and Ravindra K. Ahuja. "New scaling algorithms for the assignment and minimum mean cycle problems." In: *Math. Program.* 54 (1992), pp. 41–56. DOI: 10.1007/BF01586040.

[OR07] Rafail Ostrovsky and Yuval Rabani. "Low distortion embeddings for edit distance." In: *J. ACM* 54.5 (2007), p. 23. DOI: 10.1145/1284320.1284322.

[Pac+18] Jakub Pachocki, Liam Roditty, Aaron Sidford, Roei Tov, and Virginia Vassilevska Williams. "Approximating Cycles in Directed Graphs: Fast Algorithms for Girth and Roundtrip Spanners." In: *SODA*. SIAM, 2018, pp. 1374–1392. DOI: 10.1137/1.9781611975031.91.

[Pel02] Andrzej Pelc. "Searching games with errors - fifty years of coping with liars." In: *Theor. Comput. Sci.* 270.1-2 (2002), pp. 71–109. DOI: 10.1016/S0304-3975(01)00303-6.

[Pel89] Andrzej Pelc. "Searching with Known Error Probability." In: *Theor. Comput. Sci.* 63.2 (1989), pp. 185–202. DOI: 10.1016/0304-3975(89)90077-7.

[Pis99] David Pisinger. "Linear Time Algorithms for Knapsack Problems with Bounded Weights." In: *J. Algorithms* 33.1 (1999), pp. 1–14. DOI: 10.1006/jagm.1999.1034.

[PRW21] Adam Polak, Lars Rohwedder, and Karol Wegrzycki. "Knapsack and Subset Sum with Small Items." In: *ICALP*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 106:1–106:19. DOI: 10.4230/LIPIcs.ICALP.2021.106.

[Ram96] Rajeev Raman. "Priority Queues: Small, Monotone and Trans-dichotomous." In: *ESA*. Vol. 1136. Lecture Notes in Computer Science. Springer, 1996, pp. 121–137. DOI: 10.1007/3-540-61680-2_51.

[Ram97] Rajeev Raman. "Recent results on the single-source shortest paths problem." In: *SIGACT News* 28.2 (1997), pp. 81–87. DOI: 10.1145/261342.261352.

[Rhe15] Donguk Rhee. *Faster fully polynomial approximation schemes for knapsack problem*. 2015. URL: hdl.handle.net/1721.1/98564.

[San05] Piotr Sankowski. "Shortest Paths in Matrix Multiplication Time." In: *ESA*. Vol. 3669. Lecture Notes in Computer Science. Springer, 2005, pp. 770–778. DOI: 10.1007/11561071_68.

[Sei95] Raimund Seidel. "On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs." In: *J. Comput. Syst. Sci.* 51.3 (1995), pp. 400–403. DOI: 10.1006/jcss.1995.1078.

[Sel74] Peter H. Sellers. "On the Theory and Computation of Evolutionary Distances." In: *SIAM Journal on Applied Mathematics* 26.4 (1974), pp. 787–793. DOI: 10.1137/0126070.

[Shi55] Alfonso Shimbel. "Structure in Communication Nets." In: *Symposium on Information Networks*. Polytechnic Press of the Polytechnic Institute of Brooklyn, 1955, pp. 199–203.

[Ste13] Ernst Steinitz. "Bedingt konvergente Reihen und konvexe Systeme." In: *Journal für die reine und angewandte Mathematik* 143 (1913), pp. 128–176. DOI: 10.1515/crll.1913.143.128. URL: http://eudml.org/doc/149403.

[Tam09] Arie Tamir. "New pseudopolynomial complexity bounds for the bounded and other integer Knapsack related problems." In: *Oper. Res. Lett.* 37.5 (2009), pp. 303–306. DOI: 10.1016/j.orl.2009.05.003.

[Tar72] Robert Endre Tarjan. "Depth-First Search and Linear Graph Algorithms." In: *SIAM J. Comput.* 1.2 (1972), pp. 146–160. DOI: 10.1137/0201010.

[Tho00] Mikkel Thorup. "On RAM Priority Queues." In: *SIAM J. Comput.* 30.1 (2000), pp. 86–109. DOI: 10.1137/S0097539795288246.

[Tho04] Mikkel Thorup. “Integer priority queues with decrease key in constant time and the single source shortest paths problem.” In: *J. Comput. Syst. Sci.* 69.3 (2004), pp. 330–353. DOI: 10.1016/j.jcss.2004.04.003.

[Ukk85] Esko Ukkonen. “Finding Approximate Patterns in Strings.” In: *J. Algorithms* 6.1 (1985), pp. 132–137. DOI: 10.1016/0196-6774(85)90023-9.

[Vin68] T. K. Vintsyuk. “Speech discrimination by dynamic programming.” In: *Cybernetics* 4.1 (1968). Russian Kibernetika 4(1):81-88 (1968), pp. 52–57.

[WC22] Xiaoyu Wu and Lin Chen. “Improved Approximation Schemes for (Un-)Bounded Subset-Sum and Partition.” In: *CoRR* abs/2212.02883 (2022). DOI: 10.48550/arXiv.2212.02883.

[WF74] Robert A. Wagner and Michael J. Fischer. “The String-to-String Correction Problem.” In: *J. ACM* 21.1 (1974), pp. 168–173. DOI: 10.1145/321796.321811.

[Wil18] R. Ryan Williams. “Faster All-Pairs Shortest Paths via Circuit Complexity.” In: *SIAM J. Comput.* 47.5 (2018), pp. 1965–1985. DOI: 10.1137/15M1024524.

[Wil64] John W. J. Williams. “Algorithm 232 – Heapsort.” In: *Communications of the ACM* 7.6 (1964), pp. 347–349. DOI: 10.1145/512274.512284.

[WW18] Virginia Vassilevska Williams and R. Ryan Williams. “Subcubic Equivalences Between Path, Matrix, and Triangle Problems.” In: *J. ACM* 65.5 (2018), 27:1–27:38. DOI: 10.1145/3186893.