University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Karol Węgrzycki

# Provably Optimal Dynamic Programming

*PhD disseratation*

*Supervisor:*

dr hab. Marcin Mucha
Institute of Informatics
University of Warsaw

December 30, 2019

*Author's declaration:*
I hereby declare that this dissertation is my own work.

December 30, 2019 .....
*date* *Karol Węgrzycki*

*Supervisor's declaration:*
The dissertation is ready to be reviewed.

December 30, 2019 .....
*date* *dr hab. Marcin Mucha*

# Abstract

In this thesis we study an application of *dynamic programming* technique to graph problems and approximation algorithms. We improve upon state-of-the-art algorithms for All-Nodes Shortest Cycles, distance oracles, approximate algorithm for Partition, weak approximation for Subset Sum, and others.

We also present equivalence classes for certain problems, that admit algorithms based on dynamic programming. Namely:

- $(\min, +)$-convolution and knapsack problem,
- $(\min, \max)$-convolution and strongly polynomial approximate $(\min, \max)$-convolution,
- $(\min, \max)$-product and strongly polynomial approximate all-pairs shortest path.

**Keywords**: $(\min, +)$-convolution; tropical products; knapsack problem; dynamic programming; fine-grained complexity; strongly-polynomial algorithms; All-Pairs Shortest Path; graph algorithms;

**2012 ACM Subject Classification**: **Theory of computation → Dynamic programming**; Complexity classes; Problems, reductions and completeness;

# Streszczenie

W rozprawie przedstawiamy nowe techniki analizy algorytmów opartych na programowaniu dynamicznym. Używamy ich do rozwiązywania problemów na grafach i przyśpieszeniu wybranych algorytmów aproksymacyjnych. Zaproponowane przez nas metody pozwalają na usprawnienie obecnie znanych algorytmów dla znajdywania najkrótszych cykli, wyroczni odległości, problemów aproksymacyjnych związanych z problemem plecakowym i innych.

W rozprawie proponujemy także klasy równoważności dla wybranych problemów, które mają efektywne algorytmy oparte na programowaniu dynamicznym. W szczególności:

- $(\min, +)$-konwolucji i problemu plecakowego,
- $(\min, \max)$-konwolucji i silnie wielomianowej aproksymacji dla $(\min, +)$-konwolucji,
- $(\min, \max)$-produktu i silnie wielomianowej aproksymacji dla znajdywania najkrótszych ścieżek w grafie.

**Słowa Kluczowe**: $(\min, +)$-konwolucja; produkty tropikalne; problem plecakowy; programowanie dynamiczne; dokładna złożoność; algorytmy silnie wielomianowe; problem najkrótszych ścieżek; algorytmy grafowe;

**Tytuł rozprawy w języku polskim**: Programowanie Dynamiczne z Gwarancjami

# Acknowledgment

*In my life, I benefited tremendously from teachers and teachings of all kinds. I have met a handful of people who spend their time to pass along what they have learned. My supervisor Marcin Mucha is surely one of them. I learned from him more than I can count, but most importantly he taught me how to enjoy doing research. Thank you!*

*Teaching can also be very successfully done by books and articles. There are two authors that unwittingly inspired and taught me the most: Karl Bringmann and Jesper Nederlof.*

*I enjoy doing research – the rare thrill of excitement about a new discovery recompenses the hours of tedious work. I was fortunate enough to meet and learn from yet unmentioned friends who share the passion of doing research with me: Marek Cygan, Marvin Künnemann, Piotr Sankowski, Michał Włodarczyk with whom I also discovered results that comprise this thesis. Indeed, I feel very flattered that you chose to work and share your knowledge with me.*

---

*Warsaw, Poland* *Karol Węgrzycki*

I was financially supported by the following institutions: Grants 2016/21/N/ST6/01468 and 2018/28/T/ST6/00084 of the Polish National Science Center; Project TOTAL that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 677651); Project PAAl-POC supported by European Research Council (ERC) StG (grant agreement No 680912); Project PAAl supported by European Research Council (ERC) (grant agreement No 259515); Project MULTIPLEX financed by European Commission through FET-Proactive (grant agreement No 317532); Simons Institute for the Theory of Computing, Berkeley, University of California; Institute of Informatics of Faculty of Mathematics, Informatics, and Mechanics of University of Warsaw.

# Articles comprising this thesis

The preliminary version of content in this thesis is published in the following conference proceedings:

- *Improved Distance Queries and Cycle Counting by Frobenius Normal Form,* joint work with Piotr Sankowski, published at **STACS 2017** [130].
- *On Problems Equivalent to (min, +)-Convolution*, joint work with Marek Cygan, Marcin Mucha, Michał Włodarczyk, published at **ICALP 2017** [50].
- *A Subquadratic Approximation Scheme for Partition*, joint work with Marcin Mucha, Michał Włodarczyk, published at **SODA 2019** [117].
- *Approximating APSP without Scaling: Equivalence of Approximate Min-Plus and Exact Min-Max*, joint work with Karl Bringmann and Marvin Künnemann, published at **STOC 2019** [29].

The complete versions of selected articles are published in journal:

- *Improved Distance Queries and Cycle Counting by Frobenius Normal Form,* joint work with Piotr Sankowski, invited to the Special Issue on Theoretical Aspects of Computer Science (STACS 2017), published in **Theory of Computing Systems** [131].
- *On Problems Equivalent to (min, +)-Convolution*, joint work with Marek Cygan, Marcin Mucha, Michał Włodarczyk, published in **ACM Transactions on Algorithms** [51].

# Foreword

In here, we explain the relationship between the results in this thesis and ascertain its coherence. Initially, the main motivation to work on this thesis was the following open problem:

> *Can we compute All-Pairs Shortest Paths (APSP) for directed, unweighted graphs in time required to multiply matrices*[1] *?*

For this problem, Zwick [170] proposed an algorithm running in $\widetilde{\mathcal{O}}(n^{2.575})$ time and the question is if we can improve it. In Chapter 3 (joint work with Piotr Sankowski), we tried to use an efficient algorithm for computing the Frobenius normal form to solve this problem. It resulted in a fast algorithm for distance oracles, improved algorithms for All-Nodes Shortest Cycles (ANSC) on directed unweighted graphs, and others. However, no improvement for APSP was within our reach.

Frustrated with lack of progress, we started to seek evidence that the answer to the open problem may actually be *no*. At the time, we were particularly interested in the *APSP hypothesis*, i.e., a conjecture that no $\mathcal{O}(n^{3-\delta})$ algorithm for weighted APSP is possible. In Chapter 3, we extensively used $\mathcal{O}(n \log n)$ algorithm for *Fast Fourier Transform* (FFT) (i.e., convolution in $(+, \cdot)$-ring). We wanted to somehow apply FFT to the APSP hypothesis, hence we needed a way to generalize the convolution to the weighted case, i.e., convolution in the $(\min, +)$-semiring. Chapter 5 emerged by a scrupulous research on Min-Plus Convolution (joint work with Marek Cygan, Marcin Mucha and Michał Włodarczyk).

Then, with Marcin and Michał, we discovered that techniques developed for approximate APSP by Zwick [170] give a $\widetilde{\mathcal{O}}(\frac{n}{\varepsilon} \log W)$ approximation for Min-Plus Convolution. Since, we earlier showed that Min-Plus Convolution is equivalent to Knapsack, it seemed natural to work on approximation for Knapsack-type problems. This research resulted in Chapter 4.

[1] Currently, one can multiply matrices in $\mathcal{O}(n^{\omega}) \leq \mathcal{O}(n^{2.373})$ time [106]

Approximation for Knapsack has a long line of improvements. For some reasons unclear (to us) at that time, researchers in the late 70' were averse to weakly polynomial approximation algorithms. To our understanding, the reason was that the algorithms with $\log W$ factors do not technically satisfy the definition of *fully polynomial time approximation scheme*. This old definition made us ask the following question:

*Can we give an approximation algorithm for Min-Plus Convolution with no dependence on $W$ (even logarithmic)?*

Initially, we hoped to get an optimal $\widetilde{\mathcal{O}}(n+1/\varepsilon^2)$ algorithm for Knapsack (for which we proved a conditional lower bound, see Chapter 5). What we discovered in Chapter 6 (joint work with Karl Bringmann and Marvin Künnemann) was even more interesting. First, we gave $\widetilde{\mathcal{O}}(n^{3/2}/\sqrt{\varepsilon})$ approximation for Min-Plus Convolution and almost immediately it turned out, that the reasoning also applies to APSP. It improved (for large enough weights) an upper bound for approximation of APSP due to Zwick [170]. This way, we circled back to the original question. The question that initiated this journey and connects all results in this thesis in unexpected cycle.

# Contents

# Part I

# Overview

# Chapter 1

# Introduction

> There is nothing wrong with trying to prove that P=NP by developing a polynomial-time algorithm for an NP-complete problem. The point is that without an NP-completeness proof we would be trying the same thing without knowing it!
>
> *Christos H. Papadimitriou*

Designing an algorithm for a given problem is a laborious endeavour. Sometimes, it demands abundance of attention and vast quantities of creative acts. Despite of all that hard work, usually at the end of this process we still are not sure if our algorithm can be improved. The basic goal of complexity theory is to answer a simple question: *can our algorithm be further improved*?

To some extent, this question has been answered by the P≠NP hypothesis. However, for many problems, even the naive approach leads to a polynomial algorithm, and the P≠NP hypothesis does not seem to be particularly useful for proving polynomial lower bounds. Inspired by the statement by Christos H. Papadimitriou in the epigraph, our goal in this thesis is to show that there really are not that many hard problems for natural problems in P.

In particular, we focus on dynamic programming. We analyse current state-of-the-art algorithms for selected problems, that we find important and identify potential bottlenecks of dynamic programming. It results in improved upper bounds on the running time. Additionally, we prove that any improvement to our algorithms is *unlikely*, i.e., refined algorithm would result in a major, unexpected breakthrough in computational complexity.

**Outline:** The thesis is divided into 4 parts. In first part, we give an overview of the results of the thesis and present related work. In the second part, we present faster dynamic programming algorithms for selected problems. In the third part, we study equivalence classes for selected problems

that admit efficient dynamic programming algorithms. In the last part, we summarize the results in the thesis and give possible future direction.

The rest of this Chapter is divided into 4 independent Sections. In Section 1.1 we discuss improved dynamic programming approach for selected graph problems. We explore fast matrix multiplication and use Frobenius normal form. In Section 1.2 we give subquadratic approximation scheme for Partition. We connect pseudo-polynomial algorithm for Subset Sum that is based on dynamic programming and algorithms based on additive combinatorics. In Section 1.3 we give an overview of the field *hardness in P* and introduce the class of problems that are equivalent to Min-Plus Convolution. It turns out that this class is particularly useful in proving optimality of dynamic programming for Knapsack-type problems.

One of the best known example of essence of dynamic programming is the $\mathcal{O}(n^3)$ algorithm for All-Pairs Shortest Paths (APSP). In Section 1.4 we state our result regarding strongly polynomial approximation of APSP, introduce the class of problems equivalent to the $(\min, \max)$-product and give an overview of related work.

## 1.1 Algorithmic applications of Frobenius normal form

The *All-Pairs Shortest Paths* (APSP) problem asks to find distances between all pairs of vertices in a graph. For a directed graphs with weights in $\mathbb{R}$, there is a classical $\mathcal{O}(n^3)$ time algorithm of Floyd [60] and Warshall [156]. Currently the best upper bound for this problem is due to Williams [157] who showed an $\mathcal{O}(\frac{n^3}{2^{\Omega(\log n)^{0.5}}})$ algorithm. It is asymptotically faster than $\mathcal{O}(n^3/\log^c n)$ for any $c > 0$ (see survey [37] for earlier algorithms). Showing any algorithm that would work in $\mathcal{O}(n^{3-\epsilon})$ time for some $\epsilon > 0$ is a major open problem [157].

For undirected graphs Seidel [137] presented the optimal $\widetilde{\mathcal{O}}(n^\omega)$ time algorithm, where $\omega < 2.373$ is the matrix multiplication exponent [106]. For the directed case Zwick [170] presented an $\mathcal{O}(n^{2.575})$ time algorithm that is based on the fast rectangular matrix multiplication. Moreover, if we are interested in small integer weights from the set $\{-W, \ldots, W\}$, we have an $\mathcal{O}(W^{0.68}n^{2.575})$ algorithm [170].

Because APSP in undirected graphs can be solved in $\widetilde{\mathcal{O}}(n^\omega)$, diameter, radius, shortest cycle, etc. can be determined in $\widetilde{\mathcal{O}}(n^\omega)$ time as well. It is surprising that for a directed case, where only an $\mathcal{O}(n^{2.575})$ algorithm is known for APSP, there are also $\widetilde{\mathcal{O}}(n^\omega)$ algorithms for determining these properties. After a long line of improvements Cygan, Gabow, and Sankowski [49] showed

an $\widetilde{\mathcal{O}}(Wn^\omega)$ time algorithms for finding minimum weight perfect matching, shortest cycle, diameter and radius (some of these results were already known [128]). Also, Cygan, Gabow, and Sankowski [49] showed an application of their techniques that improves upon Yuster [164] $\widetilde{\mathcal{O}}(Wn^\omega t)$ time algorithm for the following problem: *determine the set of vertices that lie on some cycle of length at most t*. Cygan, Gabow, and Sankowski [49] managed to solve this problem in $\widetilde{\mathcal{O}}(Wn^\omega)$ time using Baur-Strassen's theorem.

All of these algorithms are effective only in the case of a dense graphs. For graphs with a small number of edges there are more efficient algorithms (e.g., APSP in $\widetilde{\mathcal{O}}(|V||E|)$ time [145]).

### 1.1.1 Our results

Consider an unweighted, directed graph $G$ with diameter $D < \infty$. In Chapter 3, we introduce a framework for counting cycles and walks of given length in matrix multiplication time $\widetilde{\mathcal{O}}(n^\omega)$. The framework is based on the fast decomposition into Frobenius normal form and the Hankel matrix-vector multiplication. It allows us to solve the following problems efficiently:

- All Nodes Shortest Cycles – for every node, return the length of the shortest cycle containing it. We give an $\mathcal{O}(n^\omega)$ algorithm, that improves upon Yuster [164] $\mathcal{O}(n^{(\omega+3)/2})$ for unweighted digraphs.

- We present the following improvement over Cygan, Gabow, and Sankowski [49] for unweighted directed graphs. Let $S(c)$ denote the set of vertices lying on cycles of length $\leq c$. Cygan, Gabow, and Sankowski [49] showed that for a fixed $c \in \{1, \ldots, n\}$, the set $S(c)$ can be computed in $\widetilde{\mathcal{O}}(n^\omega)$ randomized time. Our algorithm in the same time returns all $S(1), S(2), \ldots, S(D)$ with high probability.

- We present an $\widetilde{\mathcal{O}}(n^\omega)$ algorithm for counting non-simple cycles of all lengths $\{1, \ldots, D\}$ and improve upon [12] in the special case of batch counting ([12] counts only cycles of a fixed length).

- We improve Yuster and Zwick [167], who showed that a directed graph can be preprocessed in time $\widetilde{\mathcal{O}}(n^\omega)$, so that in $\widetilde{\mathcal{O}}(n)$ query time, we can return the distance $\delta(u, v)$ between any pair of vertices. With the same preprocessing time, our algorithm in $\widetilde{\mathcal{O}}(n)$ query time, returns $D = \mathcal{O}(n)$ numbers: $w^k_{u,v}$ for $k \in \{1, \ldots, D\}$. Here, $w^k_{u,v}$ is the count of distinct walks from $u$ to $v$ of length exactly $k$. The usual distance queries [167] are easily reduced to this problem by linear scan.

- All Pairs All Walks – we show almost optimal $\widetilde{\mathcal{O}}(n^3)$ time algorithm for all walks counting problem. The problem is to return $\mathcal{O}(n^3)$ array of numbers $w^k_{u,v}$ for all $u, v \in G$ and $k \in \{1, \ldots, D\}$. We improve upon the naive $\widetilde{\mathcal{O}}(Dn^\omega)$ time algorithm.

In contrast to the other algorithms (e.g., [49, 164]), our framework counts only cycles/walks up to the graph diameter $D$.

### 1.1.2 Related work

#### Distance queries

Yuster and Zwick [167] considered weighted, directed graphs with weights in $\{-W, \ldots, W\}$. They showed an algorithm that needs $\widetilde{\mathcal{O}}(Wn^\omega)$ preprocessing time. After preprocessing each distance $\delta(u, v)$ in the graph can be computed exactly in $\mathcal{O}(n)$ query time. In the special case $W = 1$ they showed $\widetilde{\mathcal{O}}(n^\omega)$ algorithm that solves *Single Source Shortest Paths* (SSSP). This is the best known algorithm for a dense, weighted graph.

We match their bounds (up to the polylogarithmic factors) using Frobenius normal form. Next we extend that approach so it return more information about a graph in the same query/preprocessing time.

#### Counting cycles

For a given graph $G$ and an integer $k$ determining whether $G$ contains a simple cycle of length exactly $k$ is NP-hard (in particular determining whether a graph contains a Hamiltonian cycle is NP-complete). However, for a fixed $k$ this problem can be solved in $2^{\mathcal{O}(k)}|V|^\omega$ [11].

Alon, Yuster, and Zwick [12] showed that for $k \leq 7$ one can count the number of cycles of length exactly $k$ in a graph in $\widetilde{\widetilde{\mathcal{O}}}(|V|^\omega)$ time. In [166] it is shown that for any even $k$, cycles of length $k$ can be found in $\mathcal{O}(|V|^2)$ time in undirected graphs (if they contain such a cycle). Alon, Yuster, and Zwick [12] showed more methods that depend only on the number of edges in a graph. For example for odd $k$ they showed $\mathcal{O}(E^{2-\frac{2}{k+1}})$ algorithm for finding cycles of length $k$. However, for dense graphs these results are worse than Alon, Yuster, and Zwick [11].

On the other hand, to detect whether a non-simple cycle of length exactly $k$ exists one can use the folklore algorithm. It starts by taking the adjacency matrix $A$ of a graph $G$. Subsequently, in $\widetilde{\mathcal{O}}(n^\omega)$ time compute $A^k$ by repeated squaring. If $\operatorname{Tr}\left[A^k\right] > 0$ then there exists a non-simple cycle of length $k$. [1]

[1] $\operatorname{Tr}[A]$ denotes the trace of a matrix $A$, i.e., the sum of elements on the main diagonal.

Yuster [164] considered the following problem: *for every vertex in a graph find a shortest cycle that contains it.* He called this problem *All-Nodes Shortest Cycle* (ANSC). He showed a randomized algorithm that solves ANSC for undirected graphs with weights $\{1, \ldots, W\}$ in $\widetilde{\mathcal{O}}(\sqrt{W}n^{(\omega+3)/2})$ time. He noted that for simple digraphs (directed graphs with no anti-parallel edges) it reduces to All-Pairs Shortest Path problem. The fastest known APSP algorithm for unweighted, directed graphs runs in $\mathcal{O}(n^{2.575})$ due to [170]. Here, we show how to solve ANSC in $\widetilde{\mathcal{O}}(n^{\omega})$ for general, unweighted, directed graphs. Unfortunately, our techniques allow us only to find the length of such a cycle. But we can return the set of points, that lie on some cycle of a given length. Independently to our work Agarwal and Ramachandran [9] proved that ANSC can be solved in $\widetilde{\mathcal{O}}(n^{\omega})$ for unweighted, undirected graphs using a completely different technique.

Yuster [164] also considered following problem: *given a graph and an integer $t$. Let $S(k)$ denote the set of all vertices lying in a cycle of length $\leq k$. Determine $S(t)$.* He considered directed graphs with weights in $\{-W, \ldots, W\}$ and showed $\widetilde{\mathcal{O}}(Wn^{\omega}t)$ algorithm. Recently, Cygan, Gabow, and Sankowski [49] improved his algorithm. They showed that for a fixed $t \in [0, nW]$ the set $S(t)$ can be computed in $\widetilde{\mathcal{O}}(Wn^{\omega})$ randomized time. We show, that for unweighted ($W = 1$) directed graphs we can compute sets $S(1), S(2), \ldots, S(D)$ in $\widetilde{\mathcal{O}}(n^{\omega})$ time with high probability.

## 1.2 Dynamic programming & approximation

The Knapsack-type problems are among the most fundamental optimization challenges. These problems have been studied for more than a century already, as their origins can be traced back to the 1897's paper by Mathews [112].

The Knapsack problem is defined as follows:

**Definition 1.2.1** (Knapsack)**.** *Given a set of $n$ items $E_n = \{1, \ldots, n\}$, with item $j$ having a positive integer weight $w_j$ and value $v_j$, together with knapsack capacity $t$. Select a subset of items $E \subseteq E_n$, such that the corresponding total weight $w(E) = \sum_{i \in E} w_i$ does not exceed the capacity $t$ and the total value $v(E) = \sum_{i \in E} v_i$ is maximized.*

Knapsack is one of the 21 problems featured in Karp's list of NP-complete problems [96]. We also study the case where we are allowed to take each element multiple times, called Unbounded Knapsack. Let $\Sigma(S)$ denote the sum of elements of $S$. Subset Sum is defined as follows:

**Definition 1.2.2** (Subset Sum)**.** *Given a set $S \subset \mathbb{N}$ of $n$ numbers (sometimes referred to as items) and an integer $t$, find a subset $S' \subseteq S$ with maximal $\Sigma(S')$ that does not exceed $t$.*

Subset Sum is a special case of Knapsack, where item weights are equal to item values. This problem is NP-hard as well. In fact, it remains NP-hard even if we fix $t$ to be $\Sigma(S)/2$. This problem is called the *Number Partitioning Problem* (or Partition, as we will refer to it):

**Definition 1.2.3** (Partition)**.** *Given a set $S \subset \mathbb{N}$ of $n$ numbers, find a subset $S' \subseteq S$ with maximal $\Sigma(S')$ not exceeding $\Sigma(S)/2$.*

The practical applications of Partition problem range from scheduling [94] to minimization of circuits sizes, cryptography [113], or even game theory [84, 114]. The decision version of this problem is sometimes humorously referred to as "the easiest NP-complete problem" [84]. We will demonstrate that there is a grain of truth in this claim.

All the aforementioned problems are weakly NP-hard and admit pseudo-polynomial time algorithms. The first such an algorithm for the Knapsack was proposed by Bellman [21] and runs in time $\mathcal{O}(nt)$. This bound was improved for the Subset Sum [102] and the current best (randomized) time complexity for this problem is $\widetilde{\mathcal{O}}(n+t)$, due to Bringmann [25] (for more on these and related results see Section 4.1). The strong dependence on $t$ in all of these algorithms makes them impractical for a large $t$ (note that $t$ can be exponentially larger than the size of the input). This dependence has been shown necessary as an $\mathcal{O}\big(\text{poly}(n)t^{0.99}\big)$ algorithm for the Subset Sum would contradict both the SETH [3] and the SetCover conjecture [47].

One possible approach to avoid the dependence on $t$ is to settle for approximate solutions. The notion of approximate solution we focus on in this paper is that of a *Polynomial Time Approximation Scheme* (PTAS). A PTAS for a maximization problem is an algorithm that, given an instance of size $n$ and a parameter $\varepsilon > 0$, returns a solution with value $S$, such that $\text{OPT}(1-\varepsilon) \leq S \leq \text{OPT}$. It also needs to run in time polynomial in $n$, but not necessarily in $1/\varepsilon$ (so, e.g., we allow time complexities like $\mathcal{O}(n^{1/\varepsilon})$). A PTAS is a *Fully Polynomial Time Approximation Scheme* (FPTAS) if it runs in time polynomial in both $n$ and $1/\varepsilon$. Equivalently, one can require the running time to be polynomial in $(n + 1/\varepsilon)$. For example, $\mathcal{O}(n^2/\varepsilon^4) = \mathcal{O}((n + 1/\varepsilon)^6)$.

The first approximation scheme for Knapsack (as well as Subset Sum and Partition as special cases) dates back to 1975 and is due to Ibarra and Kim [89]. Its running time is $\mathcal{O}(n/\varepsilon^2)$. After a long line of improvements [74, 75, 95, 98, 99, 105], the current best algorithms for each problem are: the $\mathcal{O}(\min\{n/\varepsilon, n+1/\varepsilon^2\})$ algorithm for Partition due to [76], the $\mathcal{O}(\min\{n/\varepsilon, n+$

$1/\varepsilon^2 \log(1/\varepsilon)\}$) algorithm for Subset Sum due to [100] and, a very recent $\widetilde{\mathcal{O}}(n + 1/\varepsilon^{9/4})$ for Knapsack due to Jin [93].

Observe that all of these algorithms work in $\Omega((n + 1/\varepsilon)^2)$ time. In fact, we are not aware of the existence of any FPTAS for an NP-hard problem working in time $\mathcal{O}((n + 1/\varepsilon)^{2-\delta})$.

**Open Question 1.2.1.** *Can we get an $\mathcal{O}((n + 1/\varepsilon)^{2-\delta})$ FPTAS for any Knapsack-type problem (or any other NP-hard problem) for some constant $\delta > 0$ or justify that it is unlikely?*

In Chapter 4 we resolve this question positively, by presenting the first such algorithm for the Partition problem. This improves upon almost 40 years old algorithm by Gens and Levner [76]. On the other hand, in Chapter 5 we provide a conditional lower bound suggesting that similar improvement for the more general Knapsack problem is unlikely.

### 1.2.1 Our results

We design the mechanism that allows us to merge the pseudo-polynomial time algorithms for Knapsack-type problems with algorithms on dense Subset Sum instances. The most noteworthy application of these reductions is the following.

**Theorem 1.2.4.** *There is an $\widetilde{\mathcal{O}}(n+1/\varepsilon^{\frac{5}{3}})$ randomized time FPTAS for Partition.*

This improves upon the previous, bound of $\widetilde{\mathcal{O}}(n+1/\varepsilon^2)$ for this problem, due to Gens and Levner [76]. Our algorithm also generalizes to a weak $(1-\varepsilon)$-approximation for Subset Sum.[2]

**Theorem 1.2.5.** *There is a randomized weak $(1-\varepsilon)$-approximation algorithm for Subset Sum running in $\widetilde{\mathcal{O}}\left(n + 1/\varepsilon^{\frac{5}{3}}\right)$ time.*

### 1.2.2 Related work

To the best of our knowledge, the fastest approximation for Partition dates back to 1980 [76] with $\widetilde{\mathcal{O}}(\min\{n/\varepsilon, n + 1/\varepsilon^2\})$ running time[3]. The majority of later research focused on matching this running time for the Knapsack and Subset Sum. In this section we will present an overview of the history of the FPTAS for these problems.

[2]Weak approximation can break the capacity constraint by a small factor. Definition 4.1.1 specifies formally what weak $(1-\varepsilon)$-approximation for Subset Sum is.

[3]As is common for Knapsack-type problems, the $\widetilde{\mathcal{O}}$ notation hides terms polylogarithmic in $n$ and $1/\varepsilon$, but not in $t$.

Table 1.1: Brief history of FPTAS for Knapsack-type problems. Since Partition is a special case of Subset Sum, and Subset Sum is a special case of Knapsack, an algorithm for Knapsack also works for Subset Sum and Partition. We omit redundant running time factors for clarity, e.g., [100] actually runs in $\tilde{\mathcal{O}}(\min\{n/\varepsilon, n+1/\varepsilon^2\})$ time but [73, 74] gave $\mathcal{O}(n/\varepsilon)$ algorithm earlier. For a more robust history see [99, Section 4.6].

| **Year** | **Reference** | **Time** | **Problem** |
|---|---|---|---|
| 1957 | Bellman [21] [99] | $\mathcal{O}(n^2/\varepsilon)$ | Knapsack |
| 1975 | Ibarra and Kim [89] and Karp [95] | $\mathcal{O}(n/\varepsilon^2)$ | Knapsack |
| 1978 | Gens and Levner [73] | $\mathcal{O}(n/\varepsilon)$ | Subset Sum |
| 1979 | Lawler [105] | $\mathcal{O}(n+1/\varepsilon^4)$ | Knapsack |
| 1980 | Gens and Levner [76] | $\mathcal{O}(n+1/\varepsilon^2)$ | Partition |
| 1994 | Gens and Levner [75] | $\mathcal{O}(n+1/\varepsilon^3)$ | Subset Sum |
| 1997 | Kellerer, Pferschy, and Speranza [100] | $\widetilde{\mathcal{O}}(n+1/\varepsilon^2)$ | Subset Sum |
| 2018 | Chan [38] | $\widetilde{\mathcal{O}}(n+1/\varepsilon^{2.4})$ | Knapsack |
| 2019 | Jin [93] | $\widetilde{\mathcal{O}}(n+1/\varepsilon^{2.25})$ | Knapsack |
| | **Chapter 4** | $\widetilde{\mathcal{O}}(n+1/\varepsilon^{1.67})$ | Partition |

The first published FPTAS for the Knapsack is due to Ibarra and Kim [89]. This naturally gives approximations for the Subset Sum and Partition as special cases. In their approach, the items were partitioned into large and small classes. The profits are scaled down and then the problem is solved optimally with dynamic programming. Finally, the remaining empty space is filled up greedily with the small items. This algorithm has a complexity $\mathcal{O}(n/\varepsilon^2)$ and requires $\mathcal{O}(n+1/\varepsilon^3)$ space.[4] Lawler [105] proposed a different method of scaling and obtained $\mathcal{O}(n+1/\varepsilon^4)$ running time.

Later, Gens and Levner [73] and Gens and Levner [74] obtained an $\mathcal{O}(n/\varepsilon)$ algorithm for the Subset Sum based on a different technique. Then, in 1980 they proposed an even faster $\mathcal{O}(\min\{n/\varepsilon, n+1/\varepsilon^2\})$ algorithm [76] for the Partition. To the best of our knowledge this algorithm remained the best. Subsequently, Gens and Levner [75] managed to generalize their result to Subset Sum with an increase of running time and obtained $\mathcal{O}(\min\{n/\varepsilon, n+1/\varepsilon^3\})$ time and $\mathcal{O}(\min\{n/\varepsilon, n+1/\varepsilon^2\})$ space algorithm [75]. Finally, Kellerer, Pferschy, and Speranza [100] improved this algorithm for Subset Sum by giving $\mathcal{O}(\min\{n/\varepsilon, n+1/\varepsilon^2 \log(1/\varepsilon)\})$ time and $\mathcal{O}(n+1/\varepsilon)$ space algorithm. This result matched (up to the polylogarithmic factors) the running time for Partition. For the Knapsack problem Kellerer and Pferschy [98] gave

[4] In [99, Section 4.6] there are claims, that 1975 Karp [95] also gives $\mathcal{O}(n/\varepsilon^2)$ approximation for Subset Sum.

an $\mathcal{O}(n \min\{\log n, \log(1/\varepsilon)\} + 1/\varepsilon^2 \log(1/\varepsilon) \min\{n, 1/\varepsilon \log(1/\varepsilon)\})$ time algorithm (note that the exponent in the parameter $(n+1/\varepsilon)$ is 3 here) and for Unbounded Knapsack Jansen and Kraft [92] gave an $\mathcal{O}(n + 1/\varepsilon^2 \log^3(1/\varepsilon))$ time algorithm (the exponent in $(n + 1/\varepsilon)$ is 2. Very recently Chan [38] presented the currently best $\widetilde{\mathcal{O}}(n + 1/\varepsilon^{12/5})$ algorithm for the Knapsack which was subsequently improved to $\widetilde{\mathcal{O}}(n + 1/\varepsilon^{9/4})$ by Jin [93].

**More related work** The naive algorithm for Knapsack works in $\mathcal{O}^*(2^n)$ time by simply enumerating all possible subsets. Horowitz and Sahni [86] introduced the meet-in-the-middle approach and gave an exact $\mathcal{O}^*(2^{n/2})$ time and space algorithm. Schroeppel and Shamir [135] improved the space complexity of that algorithm to $\mathcal{O}^*(2^{n/4})$. Improving upon these algorithms is a major open problem. Recently Bansal et al. [19] showed an $\mathcal{O}^*(2^{0.86n})$-algorithm working in polynomial space. An interesting question (and very relevant for applications in cryptography) is how hard Knapsack type problems are for random instances. For results in this line of research see [14–16, 87].

Another interesting variant of Partition is *Equal-Subset-Sum* problem where one is given a set $S$ of integers, and the task is to decide if there exist two disjoint nonempty subsets $A, B \subseteq S$, such that $\Sigma(A) = \Sigma(B)$. Recently, Mucha et al. [116] improved upon a meet-in-the-middle approach for Equal-Subset-Sum and used techniques by Bansal et al. [19] to propose a nontrivial algorithm that works in polynomial space.

## 1.3 Fine-grained complexity of tropical convolution

In recent years, significant progress has been made in establishing bounds, conditioned on conjectures other than P≠NP. Each conjecture claims time complexity lower bounds for a different problem. The main conjectures are as follows. First, the conjecture that there is no $\mathcal{O}(n^{2-\epsilon})$ algorithm for the 3SUM problem implies hardness for problems in computational geometry [67] and dynamic algorithms [124]. Second, the conjecture that there is no algorithm $\mathcal{O}(n^{3-\varepsilon})$ for All-Pairs Shortest Path (APSP) implies the hardness of determining the graph radius and graph median and the hardness of some dynamic problems (see [159] for a survey of related results). Finally, the Strong Exponential Time Hypothesis (SETH) introduced in [90, 91] has been used extensively to prove the hardness of parametrized problems [48, 110] and has recently led to polynomial lower bounds via the intermediate Orthogonal Vec-

tors problem (see [158]). These include bounds for the Edit Distance [17], Longest Common Subsequence [1, 28], and others [159].

It is worth noting that in many cases, the results mentioned indicate not only the hardness of the problem in question but also that it is computationally equivalent to the underlying hard problem. This leads to clusters of equivalent problems being formed, each cluster corresponding to a single hardness assumption (see [159, Figure 1]).

### 1.3.1 Related work

In this thesis, we propose yet another hardness assumption: the Min-Plus Convolution conjecture.

**Definition 1.3.1** (Min-Plus Convolution)**.** *In Min-Plus Convolution problem, one is given sequences* $(a[i])_{i=0}^{n-1}$, $(b[i])_{i=0}^{n-1}$. *The task is to output sequence* $(c[i])_{i=0}^{n-1}$, *such that* $c[k] = \min_{i+j=k}(a[i] + b[j])$.

This problem has previously been used as a hardness assumption for at least two specific problems [18, 104], but to the best of our knowledge, no attempts have been made to systematically study the neighborhood of this problem in the polynomial complexity landscape.

To be more precise, in all problem definitions, we assume that the input sequences consist of integers in the range $[-W, W]$. Following the design of the APSP conjecture [160], we allow $\text{polylog}(W)$ factors in the definition of a subquadratic running time.

**Conjecture 1.3.2.** *There is no* $\mathcal{O}(n^{2-\varepsilon}\text{polylog}(W))$ *algorithm for Min-Plus Convolution when* $\varepsilon > 0$.

Let us first look at the place occupied by Min-Plus Convolution in the landscape of established hardness conjectures. Figure 1.1 shows known reductions between these conjectures and includes Min-Plus Convolution. Bremner et al. [24] showed the reduction from Min-Plus Convolution to APSP. It is also known [18] that Min-Plus Convolution can be reduced to 3SUM by using reductions [124] and [161, Proposition 3.4, Theorem 3.3] (we provide the details in the Section 5.5). Note that a reduction from 3SUM or APSP to Min-Plus Convolution would imply a reduction between 3SUM and APSP, which is a major open problem in this area of study [159]. No relation between Min-Plus Convolution and SETH or OV is known.

In this thesis, we study three broad categories of problems. The first category consists of the classic Knapsack and its variants, which we show to be essentially equivalent to Min-Plus Convolution. This is perhaps somewhat

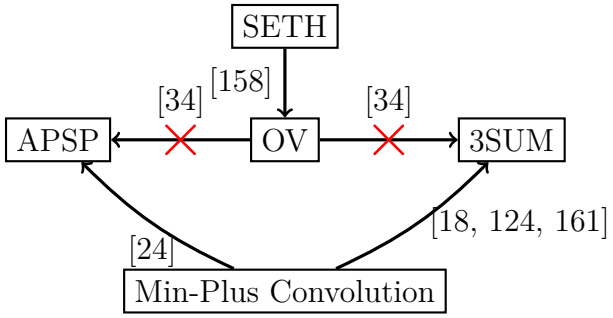



Figure 1.1: The relationship between popular conjectures. A reduction from OV to 3SUM or APSP contradicts the nondeterministic version of SETH [34, 159] (these arrows are striked out).

surprising given the recent progress of Bringmann [25] for Subset Sum, which is a special case of Knapsack. However, note that Bringmann's algorithm [25] (as well as other efficient solutions for Subset Sum) is built upon the idea of composing solutions using the $(\vee, \wedge)$-convolution, which can be implemented efficiently using a Fast Fourier Transform (FFT). The corresponding composition operation for Knapsack is Min-Plus Convolution (see Section 5.2 for details).

The second category consists of problems directly related to Min-Plus Convolution. This includes decision versions of Min-Plus Convolution and problems related to the notion of subadditivity. Any subadditive sequence $a$ with $a[0] = 0$ is an idempotent of Min-Plus Convolution; thus, it is perhaps unsurprising that these problems are equivalent to Min-Plus Convolution.

Finally, we investigate problems that have previously been shown to be related to Min-Plus Convolution and then contribute some new reductions or simplify existing ones.

### 3SUM

**Definition 1.3.3** (3SUM)**.** *In 3SUM problem, one is given sets of integers $A, B, C$, each of size $n$. The task is to decide whether there exist $a \in A$, $b \in B$, $c \in C$ such that $a + b = c$.*

The 3SUM problem is the first problem that was considered as a hardness assumption in P. It admits a simple $\mathcal{O}(n^2 \log n)$ algorithm, but the existence of an $\mathcal{O}(n^{2-\epsilon})$ algorithm remains a major open problem. The first lower bounds based on the hardness of 3SUM appeared in 1995 [67]; some other examples can be found in [20, 124, 161]. The current best algorithm for 3SUM runs in slightly subquadratic expected time $\mathcal{O}\left((n^2/\log^2 n)(\log\log n)^2\right)$ [20].

An $\mathcal{O}(n^{1.5}\text{polylog}(n))$ algorithm is possible on a nondeterministic Turing machine [34].

The 3SUM problem is known to be subquadratically equivalent to its convolution version in a randomized setting [124].

**Definition 1.3.4** (3sumConv)**.** *In 3sumConv problem, one is given sequences $a, b, c$ of integers, each of length $n$. The task is to decide whether there exist $i, j$ such that $a[i] + b[j] = c[i + j]$.*

Both problems are sometimes considered with real weights, but in this work, we restrict them to only the integer setting.

### Min-Plus Convolution

We have already defined the Min-Plus Convolution problem in Subsection 1.3.1. Note that it is equivalent (just by negating elements) to the analogous Max-Plus Convolution problem.

**Definition 1.3.5** (Max-Plus Convolution)**.** *In Max-Plus Convolution problem, one is given sequences $(a[i])_{i=0}^{n-1}$, $(b[i])_{i=0}^{n-1}$. The task is to output sequence $(c[i])_{i=0}^{n-1}$, such that $c[k] = \max_{i+j=k}(a[i] + b[j])$.*

We describe our contribution in terms of Min-Plus Convolution, as this version has been already been heavily studied. However, in the theorems and proofs, we use Max-Plus Convolution, as it is easier to work with. We also work with a decision version of the problem. Herein, we will use $a \oplus^{\max} b$ to denote the Max-Plus Convolution of two sequences $a$ and $b$.

**Definition 1.3.6** (Max-Plus Convolution UpperBound)**.** *In Max-Plus Convolution UpperBound problem, one is given sequences $(a[i])_{i=0}^{n-1}$, $(b[i])_{i=0}^{n-1}$, $(c[i])_{i=0}^{n-1}$. The task is to decide whether $c[k] \geq \max_{i+j=k}(a[i] + b[j])$ for all $k$.*

If we replace the latter condition with $c[k] \leq \max_{i+j=k}(a[i] + b[j])$, we obtain a similar problem Max-Plus Convolution LowerBound. Yet another statement of a decision version asks whether a given sequence is a self upper bound with respect to Max-Plus Convolution, i.e., if it is superadditive. From the perspective of Min-Plus Convolution, we may ask an analogous question about being subadditive (again, equivalent by negating elements). As far as we know, the computational complexity of these problems has not yet been studied.

**Definition 1.3.7** (SuperAdditivity Testing)**.** *In SuperAdditivity Testing problem, one is given a sequence $(a[i])_{i=0}^{n-1}$. The task is to decide whether $a[k] \geq \max_{i+j=k}(a[i] + a[j])$ for all $k$.*

In the standard $(+, \cdot)$ ring, convolution can be computed in $\mathcal{O}(n \log n)$ time by the FFT. A natural way to approach Min-Plus Convolution would be to design an analogue of FFT in the $(\min, +)$-semi-ring, also called the *tropical semi-ring*[5]. However, due to the lack of an inverse for the min-operation, it is unclear if such a transform exists for general sequences. When restricted to convex sequences, one can use a tropical analogue of FFT, namely, the Legendre-Fenchel transform [59], which can be performed in linear time [111]. [81] also considered sparse variants of convolutions and their connection with 3SUM.

There has been a long line of research dedicated to improving upon the $\mathcal{O}(n^2)$ algorithm for Min-Plus Convolution. Bremner et al. [24] presented an $\mathcal{O}(n^2/\log n)$ algorithm for Min-Plus Convolution, as well as a reduction from Min-Plus Convolution to APSP [24, Theorem 13]. Williams [157] developed an $\mathcal{O}(n^3/2^{\Omega(\log n)^{1/2}})$ algorithm for APSP, which can also be used to obtain an $\mathcal{O}(n^2/2^{\Omega(\log n)^{1/2}})$ algorithm for Min-Plus Convolution [39].

Truly subquadratic algorithms for Min-Plus Convolution exist for monotone increasing sequences with integer values bounded by $\mathcal{O}(n)$. Chan and Lewenstein [39] presented an $\mathcal{O}(n^{1.859})$ randomized algorithm and an $\mathcal{O}(n^{1.864})$ deterministic algorithm for this case. They exploited ideas from additive combinatorics. Bussieck et al. [32] showed that for a random input, Min-Plus Convolution can be computed in $\mathcal{O}(n \log n)$ expected and $\mathcal{O}(n^2)$ worst-case time.

Using the techniques of Carmosino et al. [34] and the reduction from Max-Plus Convolution UpperBound to 3SUM (see Section 5.5), one can construct an $\mathcal{O}\left(n^{1.5}\text{polylog}(n)\right)$ algorithm that works on nondeterministic Turing machines for Max-Plus Convolution UpperBound (see Lemma 5.4.1). This running time matches the $\mathcal{O}(n^{1.5})$ algorithm for Min-Plus Convolution in the nonuniform decision tree model [24]. This result is based on the techniques of Fredman [62, 63]. It remains unclear how to transfer these results to the word-RAM model [24].

### Knapsack

The decision versions of both Knapsack and Unbounded Knapsack problems are known to be NP-hard [72], but there are classical algorithms based on dynamic programming with a pseudo-polynomial running time $\mathcal{O}(nt)$ [21]. In fact, they are used to solve more general problems, i.e., $0/1$ KNAPSACK$^+$ and UNBOUNDED KNAPSACK$^+$, where we are asked to output answers for

[5]In this setting, Min-Plus Convolution is often called a $(\min, +)$-convolution, inf-convolution or epigraphic sum.

each $0 < t' \leq t$.

### Other problems related to Min-Plus Convolution

**Definition 1.3.8** (Tree Sparsity)**.** *In Tree Sparsity problem, one is given a rooted tree $T$ with a weight function $w : V(T) \to \mathbb{N}_{\geq 0}$, parameter $k$. The task is to find the maximum total weight of a rooted subtree of size $k$.*

The Tree Sparsity problem admits an $\mathcal{O}(nk)$ algorithm, which was at first invented for the restricted case of balanced trees [35] and then later generalized [18]. There is also a nearly linear FPTAS based on the FPTAS for Min-Plus Convolution [18]. It is known that an $\mathcal{O}(n^{2-\varepsilon})$ algorithm for Tree Sparsity entails a subquadratic algorithm for Min-Plus Convolution [18].

**Definition 1.3.9** (Maximum Consecutive Subsums Problem (MCSP))**.** *In* Maximum Consecutive Subsums Problem *(MCSP) problem, one is given a sequence $(a[i])_{i=0}^{n-1}$. The task is to output the maximum sum of $k$ consecutive elements for each $k$.*

There is a trivial $\mathcal{O}(n^2)$ algorithm for MCSP and a nearly linear FPTAS based on the FPTAS for Min-Plus Convolution [43]. To the best of our knowledge, this is the first problem to have been explicitly proven to be subquadratically equivalent to MinConv [104]. Our reduction to SuperAdditivity Testing allows us to significantly simplify the proof (see Section 5.3.1).

**Definition 1.3.10** ($l_p$-Necklace Alignment)**.** *In $l_p$-*Necklace Alignment *problem, one is given sequences $(x[i])_{i=0}^{n-1}$, $(y[i])_{i=0}^{n-1} \in [0,1)^n$ describing locations of beads on a circle. The task is to output the cost of the best alignment in the $p$-norm, i.e., $\sum_{i=0}^{n-1} d\left(x[i] + c, y[i + s \pmod n]\right)^p$, where $c \in [0,1)$ is a circular offset, $s \in \{0, \ldots, n-1\}$ is a shift, and $d$ is a distance function on a circle.*

In the $l_p$-Necklace Alignment problem, we are given two sorted sequences of real numbers $(x[i])_{i=0}^{n-1}$ and $(y[i])_{i=0}^{n-1}$ that represent two necklaces. We assume that each number in the sequence represents a point on a circle (we refer to this circle as the *necklace* and the points on it as the *beads*). The distance between beads $x_i$ and $y_j$ is defined in [24] as:

$$d(x_i, y_j) = \min\{|x_i - y_j|, (1 - |x_i - y_j|)\}$$

to represent the minimum between the clockwise and counterclockwise distances along the circular necklaces. The $l_p$-Necklace Alignment is an optimization problem where we can manipulate two parameters. The first

parameter is the offset $c$, which is the clockwise rotation of the necklace $(x[i])_{i=0}^{n-1}$ relative to the necklace $(y[i])_{i=0}^{n-1}$. The second parameter is the shift $s$, which defines the perfect matching between beads from the first and second necklaces, i.e., bead $x[i]$ matches bead $y[i+s \pmod n]$ (see [24]).

For $p = \infty$, we are interested in bounding the maximum distance between any two matched beads. The problem initially emerged for $p = 1$ during research on the geometry of musical rhythm [149]. The family of Necklace Alignment problems was systematically studied by Bremner et al. [24] for various values of $p$. For $p = 2$, they presented an $\mathcal{O}(n \log n)$ algorithm based on the FFT. For $p = \infty$, the problem was reduced to Min-Plus Convolution, which led to a slightly subquadratic algorithm. This makes $l_\infty$-Necklace Alignment a natural problem to study in the context of Min-Plus Convolution-based hardness. Interestingly, we are not able to show such hardness, which presents an intriguing open problem. Instead we reduce $l_\infty$-Necklace Alignment to a related problem.

Although it is more natural to state the problem with inputs from $[0, 1)$, we find it more convenient to work with integer sequences that describe a necklace after scaling.

Fast $o(n^2)$ algorithms for Min-Plus Convolution have also found applications in text algorithms. Moosa and Rahman [115] reduced *Indexed Permutation Matching* to Min-Plus Convolution and obtained an $o(n^2)$ algorithm. Burcsi et al. [30] used Min-Plus Convolution to obtain faster algorithms for *Jumbled Pattern Matching* and described how finding dominating pairs can be used to solve Min-Plus Convolution. Later, Burcsi et al. [31] showed that fast Min-Plus Convolution can also be used to obtain faster algorithms for a decision version of *Approximate Jumbled Pattern Matching* over binary alphabets.

### 1.3.2 Our results

Figure 1.2 illustrates the technical contributions of this thesis. The long ring of reductions on the left side of the Figure 1.2 is summarized below.

**Theorem 1.3.11.** *The following statements are equivalent:*

1. *There exists an $\mathcal{O}(n^{2-\varepsilon})$ algorithm for Max-Plus Convolution for some $\varepsilon > 0$.*
2. *There exists an $\mathcal{O}(n^{2-\varepsilon})$ algorithm for Max-Plus Convolution Upper-Bound for some $\varepsilon > 0$.*
3. *There exists an $\mathcal{O}(n^{2-\varepsilon})$ algorithm for SuperAdditivity Testing for some $\varepsilon > 0$.*

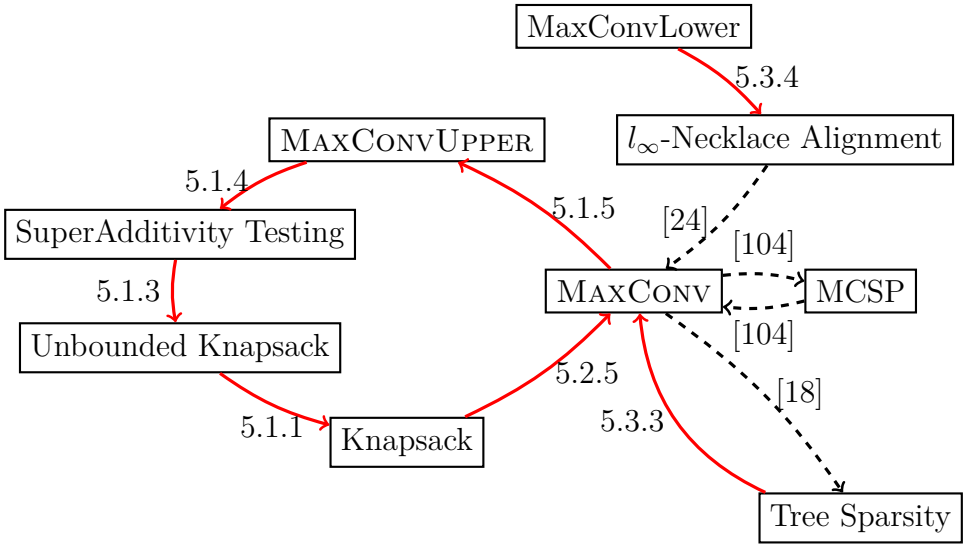



Figure 1.2: Summary of reductions in the Min-Plus Convolution complexity class. An arrow from problem $A$ to $B$ denotes a reduction from $A$ to $B$. Black dashed arrows were previously known, while red arrows correspond to new results. Numbers next to the red arrows indicate the corresponding theorems. The only randomized reduction is in the proof of Theorem 5.2.5.

4. *There exists an $\mathcal{O}((n+t)^{2-\varepsilon})$ algorithm for Unbounded Knapsack for some $\varepsilon > 0$.*

5. *There exists an $\mathcal{O}((n+t)^{2-\varepsilon})$ algorithm for Knapsack for some $\varepsilon > 0$.*

*We allow randomized algorithms.*

Additionally, we improve approximation schemes for Min-Plus Convolution. The techniques automatically yield a faster approximation for 3SUM, and Tree Sparsity. Finally, we argue why breaking the quadratic barrier for approximate Knapsack is unlikely by giving an $\Omega((n+1/\varepsilon)^{2-o(1)})$ conditional lower bound.

## 1.4 Equivalence of approximate operations in $(\min,+)$-semiring with an exact operations in $(\min,\max)$-semiring

*Scaling* is one of the most fundamental algorithmic techniques. For a problem involving weights from a range $\{1,\ldots,W\}$, the main idea of scaling is to

consider each of the $\log W$ bits one-by-one. Roughly speaking, in each phase we only consider the current bit, which simplifies the weighted problem to an unweighted problem. the previous phase. This allows to essentially focus on the unweighted case. The scaling technique was particularly successful for graph problems (e.g., [53, 65, 66, 80, 123]). For instance, a scaling-based algorithm solves maximum weighted matching in time $\mathcal{O}(m\sqrt{n}\log(nW))$ [66], which was recently improved to time $\mathcal{O}(m\sqrt{n}\log W)$ [53].

However, in some situations scaling-based algorithms may be slower than alternative approaches, since they naturally require a factor $\log W$ in the running time. In particular, in practice weights are often given as floating-point numbers, and thus $\log W$ can easily be as large as $n$, rendering most scaling-based algorithms inferior to naive approaches. For this reason as well as for the genuinely theoretical interest, research on *strongly polynomial* algorithms received major attention (e.g., [122, 134, 144, 155]). We say that an algorithm runs in strongly polynomial time if its number of arithmetic operations does not depend on $W$. For example, the fastest strongly polynomial algorithms for maximum weight matching run in time $\widetilde{\mathcal{O}}(nm)$ [57, 146]. (Here and throughout the Chapter 6 we let $\widetilde{\mathcal{O}}(T) = \mathcal{O}(T \operatorname{polylog} T)$, in particular, $\widetilde{\mathcal{O}}(n^c)$ never hides a factor $\log W$.)

The resulting challenge is to *design improved strongly polynomial algorithms* whose running times come as close as possible to the best known scaling-based algorithms, but without any $\log W$-factors. We tackle this challenge for a large class of approximation algorithms. This is achieved in part by an algorithmic framework that allows us to switch between approximate problems over the (min,+)-semiring and exact problems over the (min,max)-semiring.

**Approximating APSP, matrix products, and graph characteristics** Here, we study the following problems:

- **Shortest path problems:** The *All-Pairs Shortest Path* problem (APSP) asks to compute, given a directed graph with positive edge weights, the length of the shortest path between any two vertices.

- **Matrix products:** Given matrices $A, B \in \mathbb{R}_+^{n\times n}$, their product over the $(\oplus, \otimes)$-semiring is the matrix $C \in \mathbb{R}_+^{n\times n}$ with $C[i,j] = \bigoplus_{1\le k\le n}(A[i,k] \otimes B[k,j])$. In general, the product can be computed using $\mathcal{O}(n^3)$ semiring operations. Over the $(+,\cdot)$-ring, the problem is standard matrix multiplication and can be solved in time $\mathcal{O}(n^\omega) \le \mathcal{O}(n^{2.373})$ [106]. *Min-Plus Product* is the problem of computing the matrix product over the $(\min,+)$-semiring.

- **Graph characteristics:** Specifically, we study graph characteristics such as: *Diameter*, *Radius*, *Median*, *Minimum-Weight Triangle*, and *Minimum-Weight Cycle*.

These graph characteristics and Min-Plus Product can be reduced to APSP, and thus all of these problems can be solved in time $\mathcal{O}(n^3)$ [60, 156] and using a recent algorithm by Williams [157] in time $n^3/2^{\Omega(\sqrt{\log n})}$. Moreover, with the exception of Diameter, an $\mathcal{O}(n^{3-\delta})$-algorithm for one of these graph characteristics, or for Min-Plus Product, or for APSP would yield an $\mathcal{O}(n^{3-\delta'})$-algorithm for all of these problems [4, 160]. It is therefore conjectured that none of them can be solved in truly subcubic time [4, 160].

Zwick designed a $(1+\varepsilon)$-approximation algorithm for APSP running in time $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon}\log W)$ [170]. This yields approximation schemes with the same guarantees for Min-Plus Product and the mentioned graph characteristics [4, 128]. Zwick's running time is close to optimal[6], except that it is open whether the factor $\log W$ is necessary. To the best of our knowledge, no strongly polynomial approximation scheme is known for any of the mentioned problems. This leads to our main question:

> *Do APSP, Min-Plus Product, and the mentioned graph characteristics have strongly polynomial approximation schemes running in time $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon})$? Or at least in time $\widetilde{\mathcal{O}}(\frac{n^{3-\delta}}{\varepsilon})$ for some $\delta > 0$?*

Note that in the setting of strongly polynomial algorithms, by *time* we mean the number of arithmetic operations. However, there is also a corresponding question that considers the bit complexity. In fact, variants of our main question are reasonable and open in at least three different settings:

- *Number of arithmetic operations:* When we only count arithmetic operations, then in particular we can add/multiply two $\log W$-bit input integers in constant time. Thus, it is not clear why the running time of an algorithm should depend on $\log W$ at all. Nevertheless, Zwick's algorithm requires $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon}\log W)$ arithmetic operations. It is open whether this can be reduced to $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon})$ (or even to $\widetilde{\mathcal{O}}(\frac{n^{3-\delta}}{\varepsilon})$ for any $\delta > 0$).

- *Bit complexity with integers:* In bit complexity, an arithmetic operation on $b$-bit integers has cost $\widetilde{\mathcal{O}}(b)$. Note that the input to APSP consists of $n^2$ many $\log W$-bit integers, and suppose that we keep this number

[6] For APSP and Min-Plus Product, any $(1+\varepsilon)$-approximation can be used to compute the Boolean matrix product [45], and thus requires time $\Omega(n^\omega)$. Moreover, the dependence on $\frac{1}{\varepsilon}$ should be at least polynomial, since the hardness conjecture for APSP is stated for $W = \text{poly}(n)$ [160], and a setting of $\varepsilon = 1/W$ yields an exact algorithm.

format throughout the algorithm. Running Zwick's algorithm in this setting results in a bit complexity of $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon}\log^2 W)$, since each arithmetic operation has bit complexity $\widetilde{\mathcal{O}}(\log(nW))$. One $\log W$-factor is natural, since we operate on $\log W$-bit integers. The question thus becomes whether the *second* $\log W$-factor of Zwick's algorithm is necessary, or whether it can be improved to bit complexity $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon}\log W)$.

- *Bit complexity with floating point approximations:* One can improve upon the bit complexity of Zwick's algorithm as described above by changing the number format to floating point. Note that changing any input number by a factor in $[1, 1+\varepsilon]$ changes the resulting distances by at most $1+\varepsilon$ and thus still yields a $(1+\mathcal{O}(\varepsilon))$-approximation. We can therefore round any input integer in the range $\{1, \ldots, W\}$ to a floating point number with an $\mathcal{O}(\log \frac{1}{\varepsilon})$-bit mantissa and an $\mathcal{O}(\log\log W)$-bit exponent. We argue that this is the natural input format of Approximate APSP. In this format, arithmetic operations on input numbers have bit complexity $\widetilde{\mathcal{O}}(\log \frac{1}{\varepsilon} + \log\log W)$, and thus a factor $\log\log W$ in the bit complexity would be natural. However, implementing Zwick's algorithm in this setting yields bit complexity $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon}\log W)$. The question now becomes whether this can be improved to $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon}\log\log W)$, after converting the input numbers to floating point in time $\widetilde{\mathcal{O}}(n^2 \log W)$ (we will ignore this conversion time throughout the thesis since it is near-linear in the input size).

Note that in all three settings potentially Zwick's algorithm could be improved by a factor up to $\widetilde{\mathcal{O}}(\log W)$. We focus on the first setting, where our goal is to design algorithms whose number of arithmetic operations is independent of $W$. However, our algorithms also yield improvements in the other two settings, which we will briefly mention below.

### 1.4.1 Our results

We answer our main question affirmatively for all listed problems (for Directed APSP we need the relaxed form of the question).

For the mentioned graph characteristics, obtaining time $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon})$ is an easy exercise. Since the result is a single number, we can first compute a $\mathrm{poly}(n)$-approximation, round edge weights to obtain $W = \mathrm{poly}(n/\varepsilon)$, and then use the $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon}\log W)$-time approximation scheme as a black box.

**Theorem 1.4.1.** *Diameter, Radius, Median, Minimum-Weight Triangle, and Minimum-Weight Cycle on directed and undirected graphs have approximation schemes in strongly polynomial time $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon})$.*

For APSP restricted to *undirected* graphs, we also obtain time $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon})$. We augment an essentially standard scaling-based algorithm for APSP by contracting light edges. This is more involved than our solution for graph characteristics, and is inspired by an iterative algorithm of Tardos [144]. Similar edge contraction arguments have been used in the context of parallel algorithms for approximate APSP on undirected graphs [44, 101].

**Theorem 1.4.2.** *$(1+\varepsilon)$-Approximate Undirected APSP is in strongly polynomial time $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon})$.*

For APSP on *directed* graphs the ideas used above fail, since there are $n^2$ output numbers and we cannot contract directed edges. We obtain a *truly subcubic* strongly polynomial approximation scheme for APSP; no such algorithm was known before.

**Theorem 1.4.3.** *$(1+\varepsilon)$-Approximate Directed APSP is in strongly polynomial time $\widetilde{\mathcal{O}}(n^{\frac{\omega+3}{2}}/\varepsilon)$.*

Our approximation scheme for (directed) APSP is, in fact, a reduction from *approximate* APSP to the *exact* problem Min-Max Product, i.e., the problem of computing the matrix product over the $(\min, \max)$-semiring. This problem is closely related to the All-Pairs Bottleneck Path problem.[7] Min-Max Product and All-Pairs Bottleneck Path can be solved in time $\widetilde{\mathcal{O}}(n^{\frac{\omega+3}{2}})$ [52], which is why this term appears in our approximation scheme for APSP.

Furthermore, our reduction also works in the other direction, which yields an *equivalence* of approximation schemes for APSP and exact algorithms for Min-Max Product. In particular, for readers willing to believe that the best known running time for Min-Max Product is essentially optimal, this can be seen as a conditional lower bound for approximate APSP, showing that any improvements upon our approximation scheme in terms of the exponent of $n$ is unlikely.

**Theorem 1.4.4.** *For any $c \geq 2$, if one of the following statements is true, then all are:*

- *$(1+\varepsilon)$-Approximate Directed APSP can be solved in strongly polynomial time $\widetilde{\mathcal{O}}(n^c/\mathrm{poly}(\varepsilon))$,*
- *$(1+\varepsilon)$-Approximate Min-Plus Product can be solved in strongly polynomial time $\widetilde{\mathcal{O}}(n^c/\mathrm{poly}(\varepsilon))$,*

[7] In All-Pairs Bottleneck Path we are given a directed graph with capacities on its edges, and want to determine for all vertices $u, v$ the capacity of a single path for which a maximum amount of flow can be routed from $u$ to $v$.

- *exact Min-Max Product can be solved in strongly polynomial time* $\widetilde{\mathcal{O}}(n^c)$*,*
- *exact All-Pairs Bottleneck Path can be solved in strongly polynomial time* $\widetilde{\mathcal{O}}(n^c)$*.*

Our techniques also transfer to other problems over the (min,+)-semiring. In particular, we design a strongly polynomial $\widetilde{\mathcal{O}}(\frac{n^{3/2}}{\sqrt{\varepsilon}})$-time approximation scheme for Min-Plus Convolution (see Theorem 6.6.3). As an application, we obtain an approximation scheme with the same guarantees for the related Tree Sparsity problem (see Theorem 6.6.6) Finally, we prove an equivalence of approximating Min-Plus Convolution and exactly solving Min-Max Convolution (see Theorem 6.6.2).

Our main technical contribution is the following Sum-to-Max-Covering, which yields a framework for reducing approximate problems over the (min,+)-semiring to exact or approximate problems over the (min,max)-semiring.

**Theorem 1.4.5** (Sum-to-Max-Covering)**.** *Given vectors* $A, B \in \mathbb{R}_+^n$ *and* $\varepsilon > 0$*, in time linear in the output size we can compute vectors* $A^{(1)}, \ldots, A^{(s)}$*,* $B^{(1)}, \ldots, B^{(s)} \in \mathbb{R}_+^n$ *with* $s = \mathcal{O}((\frac{1}{\varepsilon} + \log n) \log \frac{1}{\varepsilon})$ *such that for all* $i, j \in [n]$*:*

$$A[i] + B[j] \leq \min_{\ell \in [s]} \max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \leq (1+\varepsilon)(A[i] + B[j]).$$

There are two main issues that make the proof of this statement non-trivial.

For *close pairs* $i, j$, meaning $\frac{A[i]}{B[j]} \in [\varepsilon, \frac{1}{\varepsilon}]$, the sum $A[i] + B[j]$ and the maximum $\max\{A[i], B[j]\}$ differ significantly. It is thus necessary to change the values of the vectors $A, B$. Roughly speaking, we handle this issue by splitting $A$ into vectors $A^{(\ell)}$ such that all entries $A^{(\ell)}[i], A^{(\ell)}[i']$ differ by either less than a factor $1+\varepsilon$ or by more than a factor $\text{poly}(1/\varepsilon)$. Then we can choose $B^{(\ell)}$ such that $B^{(\ell)}[j]$ is approximately $A[i] + B[j]$ for all close pairs $i, j$. This ensures that for close pairs $\max\{A^{(\ell)}[i], B^{(\ell)}[j]\}$ is approximately $A[i] + B[j]$. For details see *Close Covering* (Lemma 6.3.2).

For the *distant pairs* $i, j$, with $\frac{A[i]}{B[j]} \notin [\varepsilon, \frac{1}{\varepsilon}]$, the sum $A[i] + B[j]$ and the maximum $\max\{A[i], B[j]\}$ differ by less than a factor $1+\varepsilon$, so we do not have to change any values. However, we need to remove some entries (by setting them to $\infty$) in order to not interfere with close pairs. We show how to cover all distant pairs but no too-close pairs, via a recursive splitting into $\log n$ levels of chunks and treating boundaries between chunks by introducing several shifts of restricted areas. For details see *Distant Covering* (Lemma 6.3.3).

### 1.4.2 Related work

It is known that in general not every scaling-based algorithm can be made strongly polynomial, see, e.g., Hochbaum's work on the allocation problem [85].

**APSP and Min-Plus Product** The closest related work is by Vassilevska and Williams [152], who considered the real-valued Min-Plus Product. They proposed a method to compute the $k$ most significant bits of each entry of the Min-Plus Product in time $\mathcal{O}(2^k n^{2.687} \log n)$, in the traditional comparison-addition model of computation. This is similar to an additive $W/2^k$-approximation. However, it is incomparable to a $(1+\varepsilon)$-approximation algorithm for Min-Plus Product, since (1) the $k$ most significant bits might all be 0, in which case they do not provide a multiplicative approximation, and (2) a $(1+\varepsilon)$-approximation not necessarily allows to determine any particular bit of the result, e.g., if a number is very close to being a power of 2. Subsequently, their dependence on $n$ was improved to $\mathcal{O}(2^k n^{2.684})$[163], which was further refined to $\mathcal{O}(2^{0.96k} n^{2.687})$ and to $\mathcal{O}(2^{ck} n^{2.684})$ for some $c < 1$ [107]. Moreover, Le Gall and Nishimura [107] claim to get improved bound $\mathcal{O}(2^{ck} n^{2.684})$ for some $c < 1$, however they did not express the value of $c$ in a closed form.

For approximate APSP for real-valued graphs with weights in $[-n^{o(1)}, n^{o(1)}]$, Yuster [165] presented an additive $\varepsilon$-approximation in time $\widetilde{\mathcal{O}}(n^{\frac{\omega+3}{2}})$. More recently, among other results, Roditty and Shapira [126] gave an algorithm computing every distance $d_G(u,v)$ up to an additive error of $d_G(u,v)^p$ in time $\widetilde{\mathcal{O}}(W n^{2.575-p/(7.4-2.3p)})$. For very small $W$, this interpolates between Zwick's fastest exact algorithm and his approximation algorithm [170].

In this thesis we focus on the problem of $(1+\varepsilon)$-approximating APSP when $\varepsilon$ is close to 0. For $\varepsilon < 1$ the problem is at least as hard as Boolean matrix multiplication [45] and thus requires time $\Omega(n^\omega)$. However, there are more efficient algorithms in the regime $\varepsilon \geq 1$ for undirected graphs, using *spanners* and *distance oracles* [148].

For $k \geq 1$ a graph that $(2k-1)$-approximates all pairwise distances is called a $(2k-1)$-spanner. Thorup and Zwick [148] proved that $(2k-1)$-spanners of size $\widetilde{\mathcal{O}}(n^{1+1/k})$ can be computed in $\widetilde{\mathcal{O}}(mn^{1/k})$ time. For more recent work on graph spanners see, e.g., [2].

Our covering techniques are related to techniques proposed at [27] to compute a Min-Plus Product when a matrix has bounded differences and division of intervals in [134].

**All-Pairs Bottleneck Path and Min-Max Product** The *All-Pairs Bottleneck Path* (APBP) problem is, given an edge-weighted directed graph $G$,

to determine for all vertices $u, v$ the maximal weight $w$ such that there is a path from $u$ to $v$ using only edges of weight at least $w$. It is known that APBP is equivalent to Min-Max Product, up to lower order factors in running time. The first truly subcubic algorithm for Min-Max Product was given by Vassilevska, Williams, and Yuster [153], which was improved to time $\mathcal{O}(n^{\frac{\omega+3}{2}})$ by Duan and Pettie [52].

**Theorem 1.4.6** ([52])**.** *Given two matrices $A, B \in \mathbb{R}_+^{n\times n}$, the Min-Max Product of $A$ and $B$ can be deterministically computed in $\widetilde{\mathcal{O}}(n^{\frac{\omega+3}{2}})$.*

Shapira, Yuster, and Zwick [139] proposed an $\mathcal{O}(n^{2.575})$-time algorithm for a *vertex-weighted* variant of APBP. Duan and Ren [54] introduced the problem *All-Pairs Shortest Path for All Flows* (APSP-AF) and provided an approximation algorithm in time $\widetilde{\mathcal{O}}(n^{\frac{\omega+3}{2}}\varepsilon^{-3/2}\log W)$. They also proved an equivalence with Min-Max Product. However, in contrast to the equivalences presented in Chapter 6, their equivalence loses a factor $\log W$, and thus does not work for strongly polynomial algorithms.

APSP and APBP can be easily computed in time $\mathcal{O}(n^{2.5})$ on quantum computers [119]. Le Gall and Nishimura [107] designed the first quantum algorithm for computing Min-Max Product in time $\mathcal{O}(n^{2.473})$, and noted that every problem equivalent to APBP admits a nontrivial $\mathcal{O}(n^{2.5-\varepsilon})$-time algorithm in the quantum realm.

It is also worth mentioning that there are efficient algorithms for products in other algebraic structures, e.g., dominance product, $(+, \min)$-product, $(\min, \leq)$-product (see, e.g., [151]).

**Hardness of Approximation in P** There is a growing literature on hardness of approximation in P (see, e.g., [6, 7, 36, 40, 97, 129]), building on recent progress in fine-grained complexity theory. For readers that are willing to believe that the current algorithms for Min-Max Product are close to optimal, our equivalence of approximating APSP and exactly computing Min-Max Product is a hardness of approximation result, and in fact it is one of the first tight lower bounds for approximation algorithms for problems in P (cf. [40]).

**Convolutions** Given two vectors $a, b \in \mathbb{R}_+^n$, the convolution problem in a $(\otimes, \oplus)$ semiring is to compute $c_i = \bigoplus_k (a_k \otimes b_{i-k})$ for all $i \in [n]$. In a standard $(\cdot, +)$ ring the problem can be done in $\mathcal{O}(n \log n)$ time by using Fast Fourier Transform (FFT). The Min-Max Convolution is the problem of computing convolution in min-max semiring and Min-Plus Convolution is the problem of computing convolution in min-plus semiring.

To the best of our knowledge Kosaraju [103] was the first to give $\mathcal{O}(n^{3/2}\sqrt{\log n})$ algorithm for Min-Max Convolution (computing convolution in min-max semiring). He conjectured that his algorithm can be improved to $\widetilde{\mathcal{O}}(n)$ but so far no improvement was given.

**Theorem 1.4.7** ([103])**.** *Given two sequences $a, b \in \mathbb{R}_+^n$. The Min-Max Convolution of $a$ and $b$ can be deterministically computed in $\mathcal{O}(n^{3/2}\sqrt{\log n})$ time.*

Another related problem to Min-Max Convolution is Hamming Distance. Graf, Labib, and Uznanski [83] proposed a class of problems that are equivalent to Hamming Distance.

# Chapter 2

# Notation and preliminaries

We state our results for both directed and undirected graphs. By default, $G$ denotes a graph, $V$ the set of its vertices and $E$ set of edges. In most cases the graph is weighed with a function $w : E \to \mathbb{R}_+$. Let $d_G(u, v)$ be the length of the shortest path between vertices $u$ and $v$ in graph $G$ (note that $d_G(u, v) = \infty$ if there is no path between $u$ and $v$). When we talk about graph algorithms, $n$ denotes the number of vertices and $m$ the number of edges. For a given graph $G$ we say that $D = \max_{u,v \in V(G)} d_G(u, v)$ is a diameter of a graph. If a graph is undirected and $D < \infty$ we say that the graph is *connected* and for a directed graph with $D < \infty$ we say that it is *strongly connected*.

We consider multiplicative $(1 + \varepsilon)$-approximation algorithms and assume that $\varepsilon > 0$ is sufficiently small ($\varepsilon < 1/10$).

We use $a \oplus^{\max} b$ to denote the Max-Plus Convolution of sequences $a, b$ (see Section 5.2.2). All logarithms are base 2.

## 2.1 Fast matrix multiplication

Let $T(n)$ be the minimal number of algebraic operations needed to compute the product of $n \times n$ matrix by an $n \times n$ matrix. For now, the best known upper bound on matrix multiplication is due to Le Gall [106]:

**Theorem 2.1.1** (Le Gall [106])**.** *For every $\epsilon > 0$, $T(n) < \mathcal{O}(n^{\omega+\epsilon})$, where $\omega < 2.37287$.*

We say, that $\omega$ is the exponent of square matrix multiplication. We omit $\epsilon$ in definition and assume that we can multiply two matrices by using $\mathcal{O}(n^\omega)$ field operations.

The best lower bound for the exponent of matrix multiplication is $\omega \geq 2$. For convenience, we assume, that $\omega > 2$. Observe that $\frac{\omega+3}{2} < 2.687$.

## 2.2 Frobenius normal from

A comprehensive description of Frobenius normal form be presented in Section 3.1.1. The properties of Frobenius normal form, needed in this thesis are well known in literature [55, 141, 142].

**Theorem 2.2.1** (Storjohann [142]). *The Frobenius canonical-form of a matrix can be computed deterministically, using $\widetilde{\mathcal{O}}(n^\omega)$ field operations.*

There are also simpler probabilistic algorithms, that compute this form in expected $\widetilde{\mathcal{O}}(n^\omega)$ with high probability over small fields [56].

## 2.3 Subsums

For a finite multiset $Z \subset \mathbb{N}$ we denote its size as $|Z|$, the number of distinct elements as $||Z||$, and the sum of its elements as $\Sigma(Z)$. For a number $x$ we define $\mathrm{pow}(x)$ as the largest power of 2 not exceeding $x$. If $x < 2$ we set $\mathrm{pow}(x) = 1$. For sets $A, B \subset \mathbb{N}$ their bounded algebraic sum $A \oplus_t B$ is a set $\{a + b : a \in \{0\} \cup A,\, b \in \{0\} \cup B\} \cap [0, t]$.

**Definition 2.3.1** (Subsums). *For a finite multiset $Z \subset \mathbb{N}$ we define $\mathcal{S}(Z)_k$ as a set of all possible subset sums of $Z$ of size at most $k$, i.e., $x \in \mathcal{S}(Z)_k$ iff there exists $S' \subseteq Z$, such that $\Sigma(S') = x$ and $|S'| \leq k$. $\mathcal{S}(Z)$ is the set without the constraint on the size of the subsets, i.e., $\mathcal{S}(Z) := \mathcal{S}(Z)_\infty$. The capped version is defined as $\mathcal{S}(Z,t)_k := \mathcal{S}(Z)_k \cap [0,t]$ and $\mathcal{S}(Z,t) := \mathcal{S}(Z) \cap [0,t]$.*

*We call two multisets $Z_1, Z_2 \subset \mathbb{N}$ equivalent if $\mathcal{S}(Z_1) = \mathcal{S}(Z_2)$.*

Note that $0 \in \mathcal{S}(Z,t)_k$ for all sets $Z$ and $t, k > 0$.

## 2.4 Technical rounding lemmas

Through the thesis we extensively use the following technical lemmas about rounding integers.

**Lemma 2.4.1.** *For $k$ natural numbers $x_1, x_2, \ldots, x_k$ and positive $q, \varepsilon$ such that $q \leq \sum_{i=1}^k x_i$ and $0 < \varepsilon < 1$, it holds:*

$$\sum_{i=1}^{k} x_i \leq \frac{q\varepsilon}{k} \sum_{i=1}^{k} \left\lceil \frac{k x_i}{q\varepsilon} \right\rceil < (1+\varepsilon) \sum_{i=1}^{k} x_i.$$

*Proof.* Let $x_i = \frac{q\varepsilon}{k}c_i + d_i$ where $0 < d_i \leq \frac{q\varepsilon}{k}$ and $c_i \in \mathbb{Z}$. We have $\left\lceil \frac{kx_i}{q\varepsilon} \right\rceil = c_i + 1$. First, note that:

$$\sum_{i=1}^{k} x_i = \frac{q\varepsilon}{k} \sum_{i=1}^{k} c_i + \sum_{i=1}^{k} d_i \leq \frac{q\varepsilon}{k} \sum_{i=1}^{k} c_i + q\varepsilon = \frac{q\varepsilon}{k} \sum_{i=1}^{k} (c_i + 1),$$

what proves the left inequality. To handle the right inequality we take advantage of the assumption $\sum_{i=1}^{k} x_i \geq q$ and get:

$$(1+\epsilon)\sum_{i=1}^{k} x_i = \varepsilon \sum_{i=1}^{k} x_i + \sum_{i=1}^{k} x_i \geq q\epsilon + \sum_{i=1}^{k} x_i = q\epsilon + \frac{q\varepsilon}{k} \sum_{i=1}^{k} c_i + \sum_{i=1}^{k} d_i = \frac{q\varepsilon}{k} \sum_{i=1}^{k} (c_i + 1) + \sum_{i=1}^{k} d_i > \frac{q\varepsilon}{k} \sum_{i=1}^{k} (c_i + 1).$$

□

**Lemma 2.4.2.** *For $k$ natural numbers $x_1, x_2, \ldots, x_k$ and positive $q, \varepsilon$ such that $q \leq \sum_{i=1}^{k} x_i$ and $0 < \varepsilon < 1$, it holds:*

$$(1-\varepsilon)\sum_{i=1}^{k} x_i < \frac{q\varepsilon}{k} \sum_{i=1}^{k} \left\lfloor \frac{kx_i}{q\varepsilon} \right\rfloor \leq \sum_{i=1}^{k} x_i.$$

*Proof.* The proof is very similar to the proof of Lemma 2.4.1, however now we represent $x_i$ as $\frac{q\varepsilon}{k}c_i + d_i$ where $0 \leq d_i < \frac{q\varepsilon}{k}$ and $c_i \in \mathbb{Z}$. We have $\left\lfloor \frac{kx_i}{q\varepsilon} \right\rfloor = c_i$. The right inequality holds because:

$$\sum_{i=1}^{k} x_i = \frac{q\varepsilon}{k} \sum_{i=1}^{k} c_i + \sum_{i=1}^{k} d_i \geq \frac{q\varepsilon}{k} \sum_{i=1}^{k} c_i.$$

The left inequality can be proven as follows:

$$(1-\varepsilon)\sum_{i=1}^{k} x_i = \sum_{i=1}^{k} x_i - \varepsilon \sum_{i=1}^{k} x_i \leq \sum_{i=1}^{k} x_i - q\varepsilon \; = \left( \sum_{i=1}^{k} \frac{q\varepsilon}{k} c_i + d_i \right) - q\varepsilon < \frac{q\varepsilon}{k} \sum_{i=1}^{k} c_i.$$

□

## 2.5 Reductions

In the thesis, we present a series of results of the following form: if a problem $\mathcal{A}$ admits an algorithm with running time $T(n)$, then a problem $\mathcal{B}$ admits an algorithm with running time $T'(n)$, where function $T'$ depends on $T$ and $n$ is the length of the input. For example, we will show that $T(n) = \mathcal{O}(n^{2-\varepsilon}) \Rightarrow T'(n) = \mathcal{O}(n^{2-\varepsilon'})$. Some problems, in particular KNAPSACK, have no simple parametrization, and we allow function $T$ to take multiple arguments.

As the size of the input may increase during our reductions, we restrict ourselves to a class of functions satisfying $T(cn) = \mathcal{O}(T(n))$ for a constant $c$. This is justified, as we focus on functions of the form $T(n) = n^{\alpha}$. In some reductions, the integers in the new instance may increase to $\mathcal{O}(nW)$. In these cases, we multiply the running time by polylog$(n)$ to take into account the overhead of performing arithmetic operations.

We follow the convention of [34] and say that the decision problem $L$ admits a nondeterministic algorithm in time $T(n)$ if $L \in \text{NTIME}(T(n)) \cap \text{co-NTIME}(T(n))$.

## 2.6 Machine model and input format

Throughout this thesis, we will assume that for all approximate problems input numbers are represented in floating-point, while for all exact problems input numbers are integers represented in the usual bit representation. This choice of representation is not necessary for our new approximation algorithms (they would also work on the Word RAM with input in bit representation or on the Real RAM allowing only additions and comparisons); however, it is necessary for our equivalences between approximate and exact problems, as we discuss at the end of this section. We first describe the details of these formats as well as why this choice is well-motivated and natural.

The reader is invited to skip over the machine model details and consider an unrealistic, but significantly simpler model of computation throughout this thesis: A Real RAM model where all logical and arithmetic operations on real numbers have unit cost, including rounding operations. This model is too powerful to be a realistic model of computation [133], but considering our algorithms in this model captures the main ideas.

**Representation for exact problems** For exact problems, we assume that numbers are given in the standard binary representation. Since $W$ denotes the largest input weight, to represent one number we will need $\mathcal{O}(\log W)$ bits. In Chapters 3 and 5 we will assume that $\widetilde{\mathcal{O}}$ notation also hides polylogarith-

mic factors in $W$ for clarity of presentation (in Chapter 4 we will consider approximation problem, see next paragraph). However in Chapter 6 we explicitly focus on *strongly polynomial* approximation, therefore in Chapter 6 notation $\widetilde{\mathcal{O}}$ will never hide factors in $W$.

**Floating-point representation for approximate problems** For all approximate problems considered in this thesis, we can change every input weight by a factor $1+\frac{\varepsilon}{n}$ in a preprocessing step; this changes the result by at most a factor $1+\varepsilon$. It therefore suffices to store for each input weight $w$ its rounded logarithm $e = \lfloor \log_2 w \rfloor$, which requires only $\mathcal{O}(\log\log W)$ bits, and a $(1+\frac{\varepsilon}{n})$-approximation of $w/2^e \in [1,2]$, which requires only $\mathcal{O}(\log n \log 1/\varepsilon)$ bits. Observe that this is floating-point representation. Hence, floating-point is the natural input format for the approximate problems studied in this thesis.

The necessity for rigorous models for floating-point numbers in theoretical computer science was observed in [10, 23, 147]. Here we follow the format proposed by Thorup [146], except that we slightly simplify it, since we only want to represent positive reals. In floating-point representation, a positive real number is given as a pair $x = (e, f)$, where the *exponent* $e$ is a $\kappa$-bit integer and the *mantissa* $f$ is a $\gamma$-bit string $f_1, \ldots, f_\gamma$. The pair $x$ represents the real number

$$2^e \cdot \Big(1 + \sum_{i=1}^{\gamma} f_i/2^i\Big).$$

Here $\gamma, \kappa$ are parameters of the model. Moreover, we assume that all arithmetic operations on floating-point numbers can be performed in constant time.

For all approximate problems considered in this thesis, we assume the input weights to be given in floating-point format. In particular, if the input weights are in the range $[1, W]$, we assume floating-point representation with $\Theta(\log n)$-bit mantissa and $\Theta(\log n + \log\log W)$-bit exponent. The unit-cost assumption (that all arithmetic operations on floating-point numbers take constant time) thus hides at most a factor $\widetilde{\mathcal{O}}(\log n + \log\log W)$ compared to, e.g., the complexity of performing these operations by a device operating on bits. Observe that many other formats can be efficiently converted into floating-point, and thus our algorithms also work in other settings.

Using a fixed floating-point precision introduces inherent inaccuracies when performing arithmetic operations. For simplicity of presentation, however, we shall assume that all arithmetic operations yield an exact result. For the algorithms in this thesis, it is easy to see that this assumption can

be removed by increasing the precision slightly.

**Necessity of our choice of input representation** We crucially use our choice of input formats in our equivalences of approximate Min-Plus and exact Min-Max problems (see Theorem 1.4.4, Theorem 6.6.2): In the reduction from exact Min-Max to approximate Min-Plus we need to exponentiate some numbers. In usual bit representation, this would translate $\mathcal{O}(\log n)$-bit integers to poly$(n)$-bit integers and thus not be efficient enough. However, if $m$ is an $\mathcal{O}(\log n)$-bit integer in standard bit-representation, then we can store $2^m$ in floating-point representation by storing $m$ as the exponent; the resulting floating-point number has an $\mathcal{O}(\log n)$-bit exponent (and an $\mathcal{O}(1)$-bit mantissa).

For the other direction, from approximate problems in floating-point to exact problems in bit representation, we use that for Min-Max problems we can replace input numbers by their ranks, which converts floating-point numbers to $\mathcal{O}(\log n)$-bit integers in bit representation.

Observe that in the Min-Max type problems, one can replace all input numbers by their *ranks*, i.e., their index in the sorted ordering of all input numbers. Solving the problem by the ranks, we can then infer the result. Hence, up to additional near-linear time in the input size to determine the ranks, we can assume that all input numbers are integers in the range $\{1, \ldots, \text{poly}(n)\}$, and thus all input numbers are $\mathcal{O}(\log n)$-bit integers. This is the reason why for exact problems (studied in this thesis) bit representation is the natural input format, and not floating-point. As usual for the Word RAM, we assume that each memory cell stores $\Omega(\log n)$-bit integers (when input consists of $n$-bits), and thus operations on input numbers can be performed in constant time.

# Part II

# Faster Dynamic Programming

# Chapter 3

# Dynamic programming & Frobenius normal form

In this Chapter, we introduce the framework for counting cycles and walks of given length in matrix multiplication time $\widetilde{\mathcal{O}}(n^\omega)$. The framework is based on the fast decomposition into Frobenius normal form and the Hankel matrix-vector multiplication. It allows us to solve the All-Nodes Shortest Cycles, All-Pairs All Walks problems efficiently and also gives some improvement to distance queries in unweighted graphs.

**Outline:** In Section 3.1 we give an introduction to the cyclic subspaces and show connection with the Frobenius normal form. This gives a reader a basic understanding of the techniques that we use. Then in Section 3.2 we present an algorithm that matches the current state-of-the-art algorithm for distance queries. In the remaining Sections we improve upon this algorithm and present interesting consequences. Namely, in Section 3.3 we enrich the output to the distance query oracle with some additional information. Then in Section 3.4 we apply our observation to selected graph problems. The results in this Chapter were presented at *Symposium on Theoretical Aspects of Computer Science* (STACS 2017) [130]. The full version is published in *Theory of Computing Systems* [131].

## 3.1 Introduction to cyclic subspaces and connection to Frobenius matrices

We will start with a motivational example of application of cyclic subspaces. We have a constant-recursive sequence

$$a_n + c_0 a_{n-1} + c_1 a_{n-2} + \ldots + c_{r-1} a_{n-r} = 0$$

of order $r$, where all $c_i$ are constants and initial conditions $a_0, \ldots, a_{r-1}$ are given. Perhaps, the most familiar example is the Fibonacci sequence $F_n = F_{n-1} + F_{n-2}$.

We can define the *companion matrix* of our general sequence as:

$$C = \begin{bmatrix} 0 & \ldots & 0 & -c_0 \\ 1 & \ddots & & -c_1 \\ & \ddots & 0 & \vdots \\ 0 & & 1 & -c_{r-1} \end{bmatrix}.$$

The crucial property of the matrix $C$ is that we can generate the next element of the sequence with multiplication by $C$:

$$C^T \begin{bmatrix} a_{n-r} \\ \vdots \\ a_{n-2} \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} a_{n-r+1} \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix}.$$

For example, for Fibonacci sequence we have $F_n - F_{n-1} - F_{n-2} = 0$ for $n \geq 2$, hence $c_0 = c_1 = -1$ and:

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} F_{n-2} \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix}.$$

To get a subsequent $a_{n+1}$, one can naively repeat the procedure for the output. One can also square the companion matrix and get:

$$C^T C^T \begin{bmatrix} a_{n-r} \\ \vdots \\ a_{n-2} \\ a_{n-1} \end{bmatrix} = C^T \begin{bmatrix} a_{n-r+1} \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} = \begin{bmatrix} a_{n-r+2} \\ \vdots \\ a_n \\ a_{n+1} \end{bmatrix}.$$

And analogously to get the $a_{n+k-1}$ element we need to compute the $k$-th power of the matrix $C$. In linear algebra such transformations are known as the *cyclic subspaces* generated by the vector $a$. In the next parts of this chapter, we will restrict ourself to a conclusion, that some columns of a companion matrix occur in its powers. These properties are well known in the linear algebra theory (see [56, 71] for more cyclic properties).

### 3.1.1 Consequences of Frobenius normal form

Let $\mathbb{F}$ be a commutative field. For any matrix $A \in \mathbb{F}^{n\times n}$ there exists an invertible $U$ over $\mathbb{F}$ such that:

$$U^{-1}AU = F = \begin{bmatrix} C_1 & & & & 0 \\ & C_2 & & & \\ & & C_3 & & \\ & & & \ddots & \\ 0 & & & & C_k \end{bmatrix}.$$

and $F$ is the Frobenius-canonical-form [1] of $A$. The diagonal block $C_i$ is called the companion matrix:

$$C_i = \begin{bmatrix} 0 & & \dots & 0 & -c_0 \\ 1 & 0 & & 0 & -c_1 \\ & 1 & \ddots & \vdots & -c_2 \\ & & \ddots & 0 & \vdots \\ & & & 1 & 0 & -c_{r-2} \\ 0 & & & & 1 & -c_{r-1} \end{bmatrix} \in \mathbb{F}^{r\times r}.$$

Each companion matrix corresponds to the monic polynomial $C_i(x) = x^r + c_{r-1}x^{r-1} + \ldots + c_0 \in \mathbb{F}[x]$ (similarly to the sequence example) and this polynomial is called the *minimal polynomial* of $A$. Each minimal polynomial has a property that $C_i(A) = 0$. To guarantee that matrix has only one decomposition into Frobenius normal form we require that every polynomial must divide the next one, i.e., $C_i(x)|C_{i+1}(x)$. The final list of polynomials is called the *invariant factors* of matrix $A$ [142]. Storjohann [142] proposed the deterministic algorithm to compute the Frobenius canonical-form efficiently.

**Theorem 3.1.1** (Storjohann [142])**.** *The Frobenius canonical-form of a matrix can be computed deterministically using $\widetilde{\mathcal{O}}(n^\omega)$ field operations.*

Moreover, there are also probabilistic algorithms that compute this form in expected $\widetilde{\mathcal{O}}(n^\omega)$ time over small fields [56]. In this Chapter, all algorithms are deterministic if we the upper bound on the number of distinct walks is $W$. Then, due to the time of a single field operation we need additional $\mathcal{O}(\log W)$ factor in the complexity. However, since we are mainly interested in determining if a cycle/walk of a given length exists in a graph, we can set a sufficiently small field $\mathbb{Z}_p$ ($p$ has $\mathcal{O}(\log n)$ bits). This way when algorithm returns nonzero we are sure that there exists some walk. If algorithm returns zero, then with high probability there will be no such walk.

[1] Sometimes this form is called rational-canonical form.

### 3.1.2 Cyclic subspaces

Frobenius decomposition can be used to get the desired power of a matrix (analogously to the diagonal decomposition):

$$A^k = (UFU^{-1})^k = UF(U^{-1}U)F\cdots F(U^{-1}U)FU^{-1} = UF^kU^{-1}.$$

Moreover, we will use the property that the power of block diagonal matrix $F$ is block diagonal:

$$F^k = \begin{bmatrix} C_1^k & & & & 0 \\ & C_2^k & & & \\ & & C_3^k & & \\ & & & \ddots & \\ 0 & & & & C_l^k \end{bmatrix}.$$

Now, we need a property of companion matrices that will enable us to power them efficiently.

**Definition 3.1.2** (Cyclic Property)**.** *Let $v_1, \ldots, v_n$ be the columns of a matrix $C \in \mathbb{F}^{n\times n}$. Let $v_{n+1}, \ldots, v_{2n}$ be the columns of matrix $C^{n+1}$. If, for every $1 \le k \le n$ the columns of matrix $C^k$ are $v_k, v_{k+1}, \ldots, v_{k+n}$ then the $C$ has a* ***cyclic property****.*

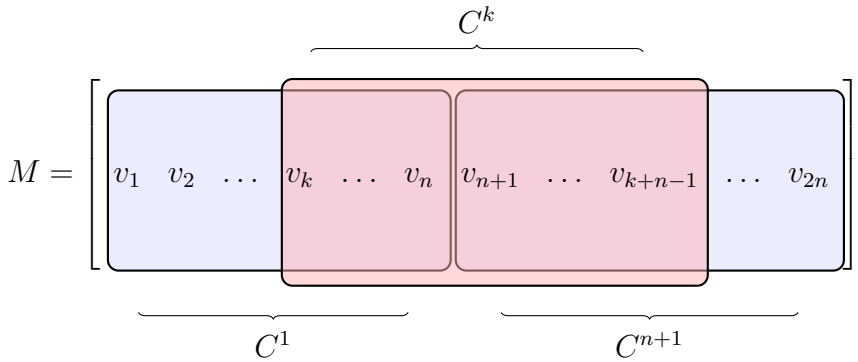



Figure 3.1: Visualisation of the cyclic property (Definition 3.1.2)

It turns out, that companion matrices have a cyclic property.

**Theorem 3.1.3** (Folklore [71], see [108] for generalization)**.** *Every companion matrix has a cyclic property.*

We will illustrate this property with an example:

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 & 4 \\ 0 & 0 & 0 & 1 & 5 \end{bmatrix}, C^2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 2 & 11 \\ 1 & 0 & 0 & 3 & 17 \\ 0 & 1 & 0 & 4 & 23 \\ 0 & 0 & 1 & 5 & 29 \end{bmatrix}, C^3 = \begin{bmatrix} 0 & 0 & 1 & 5 & 29 \\ 0 & 0 & 2 & 11 & 63 \\ 0 & 0 & 3 & 17 & 98 \\ 1 & 0 & 4 & 23 & 133 \\ 0 & 1 & 5 & 29 & 168 \end{bmatrix},$$

$$C^4 = \begin{bmatrix} 0 & 1 & 5 & 29 & 168 \\ 0 & 2 & 11 & 63 & 365 \\ 0 & 3 & 17 & 98 & 567 \\ 0 & 4 & 23 & 133 & 770 \\ 1 & 5 & 29 & 168 & 973 \end{bmatrix}, C^5 = \begin{bmatrix} 1 & 5 & 29 & 168 & 973 \\ 2 & 11 & 63 & 365 & 2114 \\ 3 & 17 & 98 & 567 & 3284 \\ 4 & 23 & 133 & 770 & 4459 \\ 5 & 29 & 168 & 973 & 5635 \end{bmatrix}.$$

The matrix $C^i$ has 4 columns identical to matrix $C^{i+1}$. $C$ has coefficients of order equal to dimension (dimension is 5 and maximum coefficient is 5). After powering to the 5th power, the coefficients can be of order $5^5$. Over a finite field $\mathbb{Z}_p$, all those coefficients will have $\mathcal{O}(\log p)$ bits.

## 3.2 Matching distance queries on directed unweighted graphs

In this section, we will present an algorithm that matches the best known upper bounds of Yuster and Zwick [167] for distance queries in directed unweighted graphs and uses Frobenius matrices.

We take the adjacency matrix $A$ of a graph $G$ (i.e., $n \times n$ matrix with $a_{u,v} = 1$ when $(u, v) \in G$ and 0 otherwise). The $k$-th power of the adjacency matrix of the graph $G$ holds the number of walks, i.e., an $a_{u,v}$ element of $A^k$ is *the count of distinct walks from $u$ to $v$ of length $k$ in the graph.*

**Observation 3.2.1** (Folklore, [42])**.** *Let $A \in \{0,1\}^{n \times n}$ be the adjacency matrix of a directed graph $G$. The $(A^k)_{u,v}$ is the number of distinct walks from $u$ to $v$ of length exactly $k$ in the graph $G$.*

Hence, the shortest path between vertices $u, v$ is the smallest $k$ such that $A^k$ has nonzero element $a_{u,v}$. This will allow us to forget about graph theory interpretation for a brief moment and focus only on finding such $k$ with algebraic tools.

In this section we will proof the following Lemma.

**Lemma 3.2.2.** *Given a matrix $A \in \mathbb{F}^{n\times n}$. There exists an algorithm that after some preprocessing, can answer queries for any given pair of indices $i, j \in \{1, \ldots, n\}$ and integer $k \in \{1, \ldots, n\}$, such that:*

- *query returns an element $(A^k)_{i,j}$,*
- *preprocessing takes $\widetilde{\mathcal{O}}(n^\omega)$ field operations and query takes $\mathcal{O}(n)$ field operations.*

*The algorithm is deterministic.*

To proof this Lemma we decompose matrix $A$ into the Frobenius normal form. Storjohann [142] showed an algorithm that returns $U$ and $F$ deterministically in $\widetilde{\mathcal{O}}(n^\omega)$ field operations (note that matrix inverse can also be computed in $\widetilde{\mathcal{O}}(n^\omega)$ field operations).

To better explain the idea, in the next section we will consider a simple case when a number of invariant factors of $A$ is exactly 1. Then in Section 3.2.2 we will show how to generalize it to multiple invariant factors.

### 3.2.1 Single invariant factor

In that situation, the matrix $F$ is a companion matrix $C \in \mathbb{F}^{n\times n}$. First, we compute the $(n+1)$-th power of the companion matrix $F^{n+1}$. This can be done by using $\widetilde{\mathcal{O}}(n^\omega)$ field operations by repeated squaring (compute $F, F^2, F^4, \ldots, F^{n+1}$ with $\mathcal{O}(\log n)$ matrix multiplications).

Let $v_1, \ldots, v_n$ be the columns of matrix $UF$ and $v_{n+1}, \ldots, v_{2n}$ be the columns of matrix $UF^{n+1}$. Note, that because matrix $F$ has a cyclic property, the columns $v_k, \ldots, v_{k+n-1}$ construct $UF^k$ (see Figure 3.2).

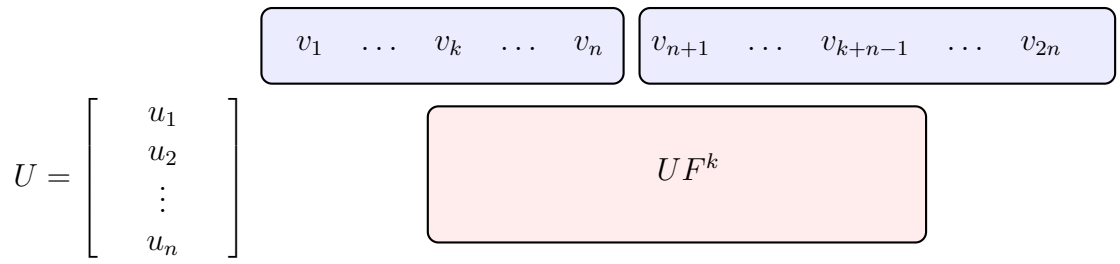



Figure 3.2: Construction of $UF^k$ from matrices $UF$ and $UF^{n+1}$

This step took just two matrix multiplications, because we need to multiply $U$ times $F$ and $F^{n+1}$. The preprocessing phase takes only $\widetilde{\mathcal{O}}(n^\omega)$ field operations.

Now, if a query asks for the number of distinct walks from vertices $u$ to $v$ of length exactly $k$ we:

- select $u$-th row of matrix $UF^k$ ($n$ numbers),
- select $v$-th column of matrix $U^{-1}$,
- multiply them in by using $\mathcal{O}(n)$ multiplications (dot product of two $n$-dimensional vectors).

This will give us the $u, v$ element of matrix $UF^kU^{-1} = A^k$. To get the length of the shortest path (i.e., the minimal $k$ such that $(A^k)_{u,v} > 0$), we will modify our matrices slightly to get the number of walks of length $\leq k$. At the end, we will use in $\tilde{\mathcal{O}}(n)$ query tim (by using binary search) and $\tilde{\mathcal{O}}(n^\omega)$ preprocessing time.

Basically, for a given $k$ we need to get the $u, v$ element of matrix $A + A^2 + \cdots + A^k$. It suffices to add consecutive columns of matrix $UF \oplus UF^{n+1} = v_1 \oplus v_2 \oplus \ldots \oplus v_{2n}$ in the following manner [2]:

$$M' = \left[ \begin{array}{cccccc} v_1 & v_1 + v_2 & v_1 + v_2 + v_3 & \ldots & \sum_{i=1}^{k} v_i & \ldots & \sum_{i=1}^{2n} v_i \end{array} \right] \in \mathbb{F}^{n \times 2n}.$$

Now, to get $A + A^2 + \cdots + A^k$ one would need to multiply $M'_{k,k+n-1}U^{-1}$ and subtract $M'_{1,n}U^{-1}$ for a balance [3].

The naive algorithm can transform matrices $U$ and $F$ to matrix $M'$ in $\mathcal{O}(n^2)$ field operations during preprocessing. During query, we will need to compute two dot products ($u$-th row of $M'_{k,k+n-1}$ times $v$-th column of $U^{-1}$ and $u$-th row of $M'_{1,n}$ times $v$-th column of $U^{-1}$) and subtract them.

We have an algorithm that for a given vertices $u, v \in G$ and integer $k \in \{1, \ldots, n\}$ can answer: *how many walks from $u$ to $v$ of length less or equal $k$ are in the graph $G$* in $\tilde{\mathcal{O}}(n)$ query time and $\tilde{\mathcal{O}}(n^\omega)$ preprocessing time.

Because the result of the query is increasing in $k$ we can use binary search. We can determine the first $k$ for which the query will answer nonzero value in $\mathcal{O}(\log n)$ tries. Hence, in $\tilde{\mathcal{O}}(n)$ we can find the length of the shortest path. This generalized query can also return the number of walks of length exactly $k$, i.e., $q(u, v, k) - q(u, v, k-1)$.

We matched the result of Yuster and Zwick [167] for unweighted graphs with a single invariant factor. In the next section, we will show how to generalize this technique for graphs with any number of invariant factors.

[2] Operation $\oplus$ denotes concatenation.

[3] $X_{a,b}$ denotes a matrix constructed by concatenating columns $x_a, x_{a+1}, \ldots, x_b$ of a matrix $X$.

### 3.2.2 Multiple invariant factors

Now, we will consider a case when $k \geq 1$, i.e., matrix $F$ has multiple invariant factors. First of all, we need to note that this generalization is not perfect and will allow to compute the number of walks of length up to $D$ (the longest distance in a graph, i.e., diameter).

In real world applications of our framework (detecting cycles, determining distance between nodes, etc.) one does not need to consider walks longer than the longest possible distance in a graph. It is natural that the diameter is considered to be a bound of an output in graph problems [5, 8, 41, 49].

#### Relation of the graph diameter and Frobenius normal form

We begin with relating the graph diameter to the Frobenius normal form. It turns out that the graph diameter is bounded by the degree of a smallest invariant factor.

**Lemma 3.2.3** ([42])**.** *Given a directed, unweighted graph $G$ with a diameter $D < \infty$. Let $\mu$ denote the degree of the smallest invariant factor (i.e., the dimension of the smallest block in the Frobenius matrix $F$) of an adjacency matrix of the graph $G$. Then $D \leq \mu$.*

This property is well known in literature [42]. We include the proof of this theorem for completeness. Note, that in Lemma 3.2.3 we assume that graph is strongly connected.

*Proof.* For a contradiction assume that $D > \mu$ and let $u, v \in G$ be vertices such that $\delta(u, v) = D$. Let $A$ be the adjacency matrix of $G$. We know, that there is the minimal polynomial of degree $\mu$ ($\mathbb{I}$ denotes the identity matrix):

$$A^{\mu} = a_0\mathbb{I} + a_1A + a_2A^2 + \ldots + a_{\mu-1}A^{\mu-1}.$$

Term $a^k_{i,j}$ denotes the $i, j$ element of the matrix $A^k$. Now, consider the elements $u, v$ of each matrix. The diameter $D > \mu$ and $\delta(u, v) = D$, so for every $k \leq \mu$ the elements $a^k_{u,v} = 0$ (because there is no walk of length less than $D$ from $u$ to $v$). Now, if we multiply the minimal polynomial by the matrix $A$ we get:

$$A^{\mu+1} = a_0A + a_1A^2 + a_2A^3 + \ldots + a_{\mu-1}A^{\mu}.$$

Hence $a^{\mu+1}_{u,v} = 0$, because every element in the sequence $a^k_{u,v} = 0$ for $k \leq \mu$. By repeating this reasoning, we get that for every $k > 0$ the element $a^k_{u,v} = 0$. So, for every achievable pair of vertices, there must be some $k \leq \mu$, such that $a^k_{u,v} \neq 0$ and diameter is bounded by $\mu$. □

The bounds of this inequality are tight. There are graphs with diameter $D = \mu$ and graphs with $\mu = n$ and arbitrary small diameter [42]. Our algorithms are able to return walks up to the length $\mu$. We use the bound on $D$ solely because it is easier to interpret diameter than the *smallest degree of the invariant factor*.

### Generalization to multiple invariant factors

Let $k$ denote the number of blocks in the Frobenius matrix $F$ and $\mu$ be the number of columns of the smallest block. To multiply the matrix $U$ by $F$ we can start by multiplying strips of matrix $U$ by appropriate blocks of $F$ and concatenate them later (see Figure 3.3).

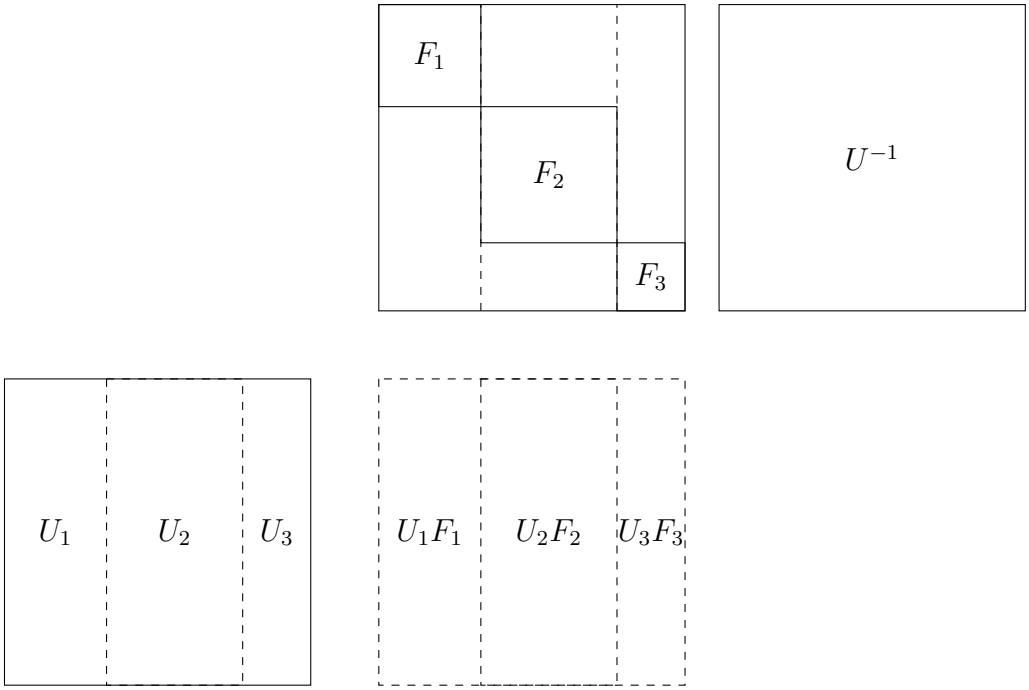



Figure 3.3: Multiplication of the $UFU^{-1}$. Example for 3 invariant factors. We know that matrix $F$ is block and consists of $F_1, F_2, F_3$. We divide matrix $U$ into strips $U_1, U_2, U_3$ that corespond to blocks of $F$. The observation is that we can compute $U_1 \cdot F_1$, $U_2 \cdot F_2$ and $U_3 \cdot F_3$ independently and concatenate them into matrix $UF$.

We start by splitting the matrix $U$ into $k$ strips with rows corresponding to the appropriate blocks of $F$ (strip $U_i$ has as many columns as block $F_i$). Then we multiply $UF$ and have $k$ strips: $U_1F_1, U_2F_2, \ldots U_kF_k$ (each

with at least $\mu$ columns). Next, we multiply $UF^\mu$ and also keep $k$ strips: $U_1F_1^\mu, U_2F_2^\mu, \ldots, U_kF_k^\mu$. Our goal is to get a data structure such that if we need $UF^k$, we can quickly choose appropriate columns and append them.

The matrix $U_iF_i$ has $l_i$ columns: $v_1, \ldots, v_{l_i}$. Because $F_i$ is a companion matrix, the $U_iF_i^\mu$ has the cyclic property (Definition 3.1.2). And the matrix $U_iF_i^\mu$ has columns: $v_\mu, \ldots, v_{\mu+l_i}$. Note, that there are some duplicate columns in $U_iF_i$ and $U_iF_i^\mu$, when $\mu < l_i$. Hence, we only need to keep columns $v_1, \ldots, v_{\mu+l_i}$ for this strip. We do this for all strips $U_1F_1, \ldots U_kF_k$ (see Figure 3.4).

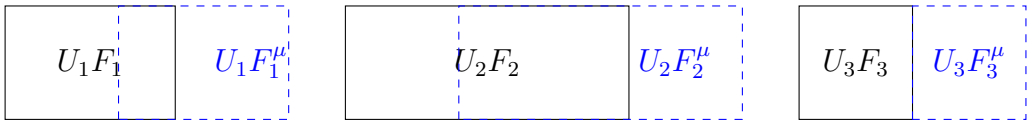



Figure 3.4: Combining strips into a single matrix. The height of the matrix in the schema is scalled down. We computed $U_1F_1$ and $U_1F_1^\mu$. Now we noted that companion matrices have a cyclic property so some of the rows in the strips are repeated. So in the single strip we can store only subsequent columns.

We are left with a matrix that has at most $2n$ columns (because $l_1 + \mu + l_2 + \mu + \ldots l_k + \mu = k\mu + \sum_{i=1}^n l_i = n + k\mu \leq 2n$). To generate it we need to power $F$ to $\mu$ and do multiplications $U \cdot F$ and $U \cdot F^\mu$. This can be computed in $\widetilde{\mathcal{O}}(n^\omega)$ field operations via fast matrix multiplication and repeated squaring.

### Queries with multiple invariant factors

When a query for the number of walks of length $k$ from node $u$ to $v$ comes, we do:

1. For each strip $i$ take the $u$-th row of $U_iF_i \oplus U_iF_i^\mu$ and concatenate them (see Figure 3.5) into vector $\bar{u}$,
2. Take $v$-th column of $U^{-1}$ matrix and denote it $\bar{v}$,
3. Return the dot product $\bar{u} \cdot \bar{v}$.

Because $l_1 + l_2 + \ldots + l_k = n$ the vector $\bar{u} \in \mathbb{F}^n$. Query needs $\mathcal{O}(n)$ field operations.

Finally, this dot product is $a_{u,v}$ element of the matrix $UF^kU^{-1}$, for a fixed $k \leq \mu$ because $\bar{u}$ is the concatenation of original vector $u$. Analogously

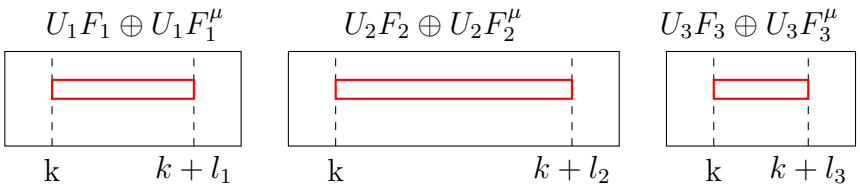



Figure 3.5: Schema of obtaining vector $\bar{u}$ (marked red) from 3 strips. We are given the row number and the power $k$ and the lenghts $l_i$ of each strip. At the end we concatenate them.

to Section 3.2.1 one can extend this result to return the number of walks of length less or equal $k$. This matches (up to the polylogarithmic factor) the result of Yuster and Zwick [167]. We will omit the details of this observation because in the next section we will extend this framework even further.

## 3.3 Almost optimal query

In the previous section, we showed how to preprocess a matrix $A$ with $\widetilde{\mathcal{O}}(n^{\omega})$ field operations in such a way that in query that uses $\mathcal{O}(n)$ field operations we can return a number $(A^k)_{i,j}$. However, in linear time $\mathcal{O}(n)$ we return only a single number. The goal of this section is to get far richer information in $\widetilde{\mathcal{O}}(n)$ query time (with extra factors from field operations).

**Theorem 3.3.1.** *Let $A \in \{0,1\}^{n\times n}$ be a matrix such that the degree of smallest invariant factor is $\mu$. There exists a deterministic algorithm that after some preprocessing can answer queries for any given $i, j \in \{1, \ldots, n\}$:*

- *Query returns $\{a_k \mid 1 \leq k \leq \mu\}$, where $a_k = (A^k)_{i,j}$ ,*
- *Preprocessing takes $\widetilde{\mathcal{O}}(n^{\omega} \log W)$ and query takes $\widetilde{\mathcal{O}}(n \log W)$ time,*

*where $W$ is an upper bound on $a_k$ for all $k \in \{1, \ldots, \mu\}$.*

Note, that this theorem has some immediate application in graph algorithms (see Section 3.4).

### 3.3.1 Hankel matrix

Now, we will focus on the proof of Theorem 3.3.1. First, we need to introduce the Hankel matrix and its properties.

$$H = \begin{pmatrix} c_1 & c_2 & \dots & c_n \\ c_2 & c_3 & \dots & c_{n+1} \\ \vdots & \vdots & & \vdots \\ c_n & c_{n+1} & \dots & c_{2n-1} \end{pmatrix}$$

Hankel matrix is defined by its first row and last column ($2n-1$ numbers define $n \times n$ Hankel matrix). The numbers from the previous row are left-shifted by one and the new number is added at the end. Hankel matrices have some similarities to Topelitz and Circulant matrices.

The basic property we need is that the product of Hankel matrix and vector can be computed in $\mathcal{O}(n \log n)$ time (see [82, 143]) even though explicitly writing the Hankel matrix as $n \times n$ matrix takes $\mathcal{O}(n^2)$ time. The algorithm takes $2n-1$ parameters that define the Hankel matrix and $n$ parameters that define the vector. The technique is based on the Fast Fourier Transformation [82, 143].

### 3.3.2 Applying Hankel matrices

To proof the Theorem 3.3.1 we will modify only the last step in Section 3.2.2. The algorithm from Section 3.2.2 concatenates the strips $U_iF_i$ and builds a single vector. Subsequently, that vector is multiplied by a column of matrix $U^{-1}$. But we can also do it in a different order: first we multiply the strip by a section of matrix $U^{-1}$ and sum the results at the end. Thus, we perform $k$ (number of strips) dot products of smaller vectors (see Figure 3.6).

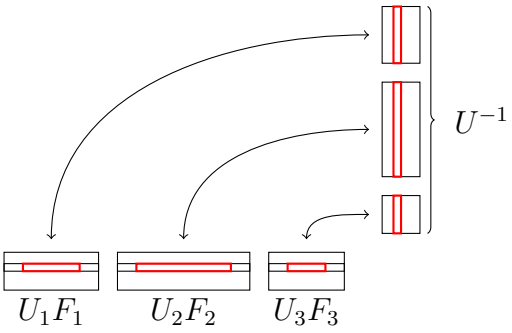



Figure 3.6: Multiplication of strips by $U^{-1}$ matrix. As you can see, matrix $U^{-1}$ can be splited into sections, that multiply only $U_iF_i$ strips.

Consider the query for a number of walks of length exactly $k$. The strips in the matrix $U^{-1}$ do not depend on $k$ (vector $(u_0, \dots, u_l)$). However, the

vector taken from $U_iF_i$ (vectors $(x_i, \ldots, x_{i+l})$) will be left shifted if we want to compute the next one.

$$\begin{matrix} (x_0 & x_1 & x_2 & \ldots & x_l) \\ (x_1 & x_2 & \ldots & x_l & x_{l+1}) \\ (x_2 & \ldots & x_l & x_{l+1} & x_{l+2}) \\ \vdots & & & & \vdots \\ (x_\mu & & \ldots & & x_{\mu+l}) \end{matrix} \times \begin{pmatrix} u_0 \\ \\ \vdots \\ \\ u_l \end{pmatrix}$$

As you can see, the subsequent rows can be written as the Hankel matrix (we need to add zeros to get a square matrix, but it will not influence asymptotic complexity since there will be at most $\mathcal{O}(n)$ of them). By using the *fast Hankel matrix-vector multiplication* we can compute $\mu$ values for every strip $i$ in time $\mathcal{O}(l_i \log l_i)$ ($l_i$ was defined as the length of $i$-th strip). At the end, we need to sum all results into a single array. Therefore, the number of operations is $\mathcal{O}\left(\sum_{i=1}^{k} l_i \log l_i\right)$. Because $\sum_{i=1}^{k} l_i = n$ the algorithm needs $\mathcal{O}(n \log n)$ field operations. This proves Theorem 3.3.1.

Here, we silently assumed that the number of walks is bounded by $W$. Note, that for large $W$, the algorithm needs to output $\mathcal{O}(n^2 \log W)$ bits and the complexity of every arithmetic operation needs to be multiplied by $\log W$. If one is only interested in the deciding if an entry of some power of adjacency matrix is nonzero, we can use a standard randomization technique to eliminate $\log W$ factors from the running time.

**Corollary 3.3.2.** *Let $A \in \{0,1\}^{n \times n}$ be a matrix such that the degree of smallest invariant factor is $\mu$. There exists an algorithm that after some preprocessing can answer queries for any given $i, j \in \{1, \ldots, n\}$:*

- *Query returns $\{a_k \mid 1 \leq k \leq \mu\}$, where $a_k = 1$ if $(A^k)_{i,j}$ is nonzero,*
- *Preprocessing takes $\widetilde{\mathcal{O}}(n^\omega)$ and query takes $\widetilde{\mathcal{O}}(n)$ time.*

*The algorithm is randomized with one-sided bounded error.*

*Proof.* At the beginning we will randomly select a prime number with $\mathcal{O}(\log n)$ bits. We can write the matrix $A$ as a polynomial:

$$\tilde{A}_{i,j} = \begin{cases} x_{i,j} & \text{if } A_{i,j} = 1 \\ 0 & \text{otherwise,} \end{cases}$$

where $x_{i,j}$ are unique variables. Now we can apply Schwartz-Zippel Lemma [136, 168]. The $\tilde{A}^k_{i,j}$ is the polynomial of degree at most $n$ (because $k \leq \mu \leq n$) and if we compute it over $\mathbb{Z}_p$ for $p = \mathcal{O}(n^2)$ we can correctly

determine if $\tilde{A}^k_{i,j}$ is a nonzero polynomial with a constant probability. We can repeat the above procedure $\mathcal{O}(\log n)$ times to get a correct result in all entries with high probability. □

## 3.4 Applications

In this section we will show how to use Theorem 3.3.1 to improve known algorithms on graphs. First we will develop a data structure that returns a number of distinct walks efficiently.

**Lemma 3.4.1.** *Let $G = (V, E)$ be a strongly connected, directed, unweighted graph with $n$ vertices and a diameter $D < \infty$. There exists an deterministic algorithm that after some preprocessing can answer queries for any given $u, v \in V$:*

- *Query returns $\{w_i \mid 1 \leq i \leq D\}$, where $w_i$ is the number of distinct walks from $u$ to $v$ of length exactly $i$,*
- *Preprocessing takes $\widetilde{\mathcal{O}}(n^\omega \log W)$ and query takes $\widetilde{\mathcal{O}}(n \log W)$ field operations.*

*where $W$ is the upper bound on $w_i$ for all $i \in \{1, \ldots, D\}$.*

*Proof.* We encode the graph $G$ as an adjacency matrix $A(G)$. We use the Theorem 3.3.1 to construct the data structure that given a query $(u, v)$ outputs $(A^k)_{u,v}$ for all $1 \leq k \leq \mu$. Finally, we use Observation 3.2.1 to note, that $(A^k)_{u,v}$ is equal to the number of distinct walks from $u$ to $v$ of length exactly $k$. Moreover we use Lemma 3.2.3 to bound the number $D \leq \mu$, so we will always output more numbers (but we can truncate them in $\mathcal{O}(n)$ time). Finally we note, that the preprocessing and query of Theorem 3.3.1 matches the statement and construction of adjacency matrix is $\mathcal{O}(n^2)$. □

This algorithm is a significant improvement over Yuster and Zwick [167]:

- One can use Lemma 3.4.1 to find the distance between $u, v$ by linearly scanning the array and returning the first $k$ such that $w_k > 0$,
- Lemma 3.4.1 can count cycles. In contrast the Yuster and Zwick [167] cannot, because the distance from $u$ to $u$ is always 0 (see Section 3.4.1),
- Lemma 3.4.1 is almost optimal, i.e., when $D = \mathcal{O}(n)$ then query will need to output $\mathcal{O}(n \log W)$ bits.

From the other hand, Lemma 3.4.1 is merely a functional improvement and it does not break the $\widetilde{\mathcal{O}}(n^\omega)$ of the *Single Source Shortest Paths* (SSSP) for dense, directed graphs.

Now we will show the application of Lemma 3.4.1. We begin with almost optimal algorithm to compute the number of all walks between all pairs of vertices. We are not aware of any other research concerning this problem.

**Definition 3.4.2** (All-Pairs All Walk problem)**.** *Given a strongly connected, directed, unweighted graph $G$ with a diameter $D < \infty$. The task is to return an array $A$, such that for every pair of vertices $u, v \in G$ and every $k \in \{1, \ldots, D\}$ an element $A[u, v, k]$ is the number of distinct walks from $u$ to $v$ of length $k$.*

The folklore solution to this needs $\mathcal{O}(Dn^\omega \log W)$ time (where $W$ is an upper bound on number of walks) and works as follows: *take the adjacency matrix $A$ of graph $G$ and save it in $A[u, v, 1]$. Then, square it to get $A^2$ and save it in $A[u, v, 2]$. Continue until you fill out complete table.* In the worst case this algorithm needs $D = \mathcal{O}(n)$ matrix multiplications, thus it needs $\mathcal{O}(Dn^\omega)$ field operations. At the first glance it is surprising that we can improve it to $\widetilde{\mathcal{O}}(n^3)$ field operations.

**Theorem 3.4.3.** *All-Pairs All Walk problem admits an $\widetilde{\mathcal{O}}(n^3 \log W)$ algorithm (where $W$ is upper bound on number of walks between every pair of vertices).*

*Proof.* We will apply the Lemma 3.4.1 algorithm. The preprocessing takes $\widetilde{\mathcal{O}}(n^\omega)$ time. Then, for every pair of vertices $u, v$ ask a query. A single query takes $\widetilde{\mathcal{O}}(n \log W)$ time. Next we will save it in the table $A[u, v]$ (query gives $D$ numbers $w_1, \ldots, w_D$, such that $w_i$ is the number of walks of length i and save it $A[u, v, i] \coloneqq w_i$).

Because there are $\mathcal{O}(n^2)$ pairs and for each pair we need $\widetilde{\mathcal{O}}(n \log W)$ time, the complexity of our solution is $\widetilde{\mathcal{O}}(n^3 \log W)$. The algorithm is almost optimal because the output in the worst case may be $\mathcal{O}(n^3 \log W)$ (we may need $\mathcal{O}(\log W)$ bits to encode a single entry in the table). □

### 3.4.1 Counting and determining the lengths of cycles

We will use Theorem 3.3.1 to solve All-Nodes Shortest Cycle (ANSC) problem efficiently.

**Definition 3.4.4** (All-Nodes Shortest Cycles [164])**.** *Given a directed, unweighted graph $G$. The problem All-Nodes Shortest Cycle asks to output for every vertex $v$ the length of the shortest cycle that contains $v$.*

**Lemma 3.4.5.** *There exists a deterministic algorithm that for a given unweighted, directed $G$ with a diameter $D < \infty$:*

- *For every vertex $u$ returns $D$ numbers: $c_u^1, c_u^2, \ldots c_u^D$, where $c_u^k$ is the number of non-simple cycles of length exactly $k$, that contain vertex $u$,*
- *Algorithm works in $\widetilde{\mathcal{O}}(n^\omega \log W)$ time (where $W$ is an upper bound on $c_u^k$).*

*Proof.* We will use Theorem 3.3.1. We start by preprocessing the graph $G$ in time $\widetilde{\mathcal{O}}(n^\omega \log W)$. Theorem 3.3.1 allows us to ask for a number of walks from $u$ to $v$ and receive $D$ numbers: $w_{u,v}^k$. So, we ask for the number of walks from vertex $u$ to the same vertex $u$. This is exactly the number of non-simple cycles of given length that contain vertex $u$.

Because we need to ask only $n$ queries (it is the number of vertices in a graph) and each query takes $\widetilde{\mathcal{O}}(n \log W)$ time we have $\widetilde{\mathcal{O}}(n^\omega \log W + n^2 \log W) = \widetilde{\mathcal{O}}(n^\omega \log W)$ algorithm.

□

If we are only interested in deciding if the numbers $c_u^i$ are nonzero, instead of Theorem 3.3.1 we can use Corollary 3.3.2. It introduces the one-sided randomization but allows us to shave $\log W$ factors in the running time.

**Corollary 3.4.6.** *There exists a randomized algorithm that for a given unweighted, directed $G$ with a diameter $D < \infty$:*

- *For every vertex $u$ returns $D$ numbers: $c_u^1, c_u^2, \ldots c_u^D$, where $c_u^k$ is 1 if there exists a non-simple cycle of length exactly $k$, that contain vertex $u$ or 0 otherwise,*
- *Algorithm works in $\widetilde{\mathcal{O}}(n^\omega)$ time with one sided bounded error.*

Now we will show how to improve upon Yuster [164] $\widetilde{\mathcal{O}}(n^{(\omega+3)/2})$ algorithm with Corollary 3.4.6.

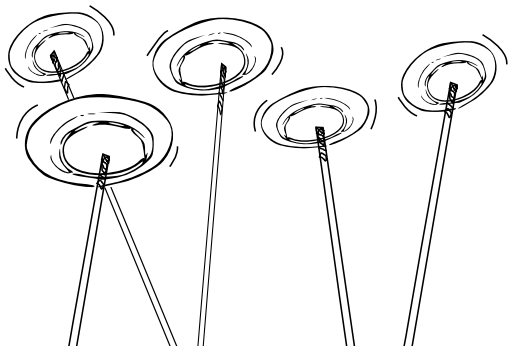
**Theorem 3.4.7.** *All-Nodes Shortest Cycles admits an $\widetilde{\mathcal{O}}(n^\omega)$ randomized time algorithm when underlying graph is strongly connected.*

*Proof.* We use Lemma 3.4.6 to compute the table $S[v]$. For every vertex we search for the first nonzero element linearly. This with high probability is the length of the shortest cycle that contains it. Because the output contains $O(n^2)$ numbers the complexity is equal to the preprocessing time $\widetilde{\mathcal{O}}(n^\omega)$. □

Also the Corollary 3.4.6 improves upon Cygan, Gabow, and Sankowski [49, Theorem 45] for unweighted graphs.

**Corollary 3.4.8.** *Given a directed, unweighted graph $G$ with a diameter $D < \infty$. Let $S(c)$ denote the set of vertices that lie in the cycle of length exactly $c$. In $\widetilde{\mathcal{O}}(n^\omega)$ time we can return the sets $S(1), \ldots, S(D)$ with constant probability of success.*

*Proof.* Similarly to the proof of Theorem 3.4.7, we can scan the output to compute the set $S(c)$ that contains all vertices that lie on some cycle of length $\leq c$. Then, by linear scan we can return the sets $S(1), \ldots, S(D)$. □



# Chapter 4

# Dynamic programming meets approximation

In this Chapter we study the time complexity of approximating Knapsack, Subset Sum, Partition, and some other related problems. The main result is an $\tilde{\mathcal{O}}(n + 1/\varepsilon^{5/3})$ time randomized FPTAS for Partition, which is derived from a certain relaxed form of a randomized FPTAS for Subset Sum. To the best of our knowledge, this is the first NP-hard problem that has been shown to admit a subquadratic time approximation scheme, i.e., one with time complexity of $\mathcal{O}((n + 1/\varepsilon)^{2-\delta})$ for some $\delta > 0$. To put these developments in context, note that a quadratic FPTAS for Partition has been known for 40 years.

Our main contribution lies in designing a mechanism that reduces an instance of Subset Sum to several simpler instances, each with some special structure, and keeps track of interactions between them. This allows us to combine techniques from approximation algorithms, pseudo-polynomial algorithms, and additive combinatorics.

**Outline:** In Section 4.1 we present the building blocks of our framework and a sketch of the approximation scheme for Partition. The main proof is divided into Sections 4.2 and 4.3. Preliminary version of results in this Chapter was presented at *Symposium on Discrete Algorithms* (SODA 2019) [117].

The drawing illustrates an art of plate spinning. It depicts a careful balance between the running time and approximation error that is preserved in this Chapter.

## 4.1 Overview of the techniques

In this Section we describe main building blocks of our framework. We also briefly discuss the recent advances in the pseudo-polynomial algorithms for Subset Sum and discuss how to use them. Then, we explain the intuition behind the trade-off we exploit and give a sketch of the main algorithm. The formal arguments are located in Section 4.3.

**Difficulties with Rounding for Subset Sum** There is a strong connection between approximation schemes and pseudo-polynomial algorithms [162]. For example, a common theme in approximating Knapsack is to reduce the range of the *values* (while keeping the *weights* intact) and then apply a pseudo-polynomial algorithm. Rounding the weights would be tricky because of the hard Knapsack constraint. In particular, if one rounds the weights down, some feasible solutions to the rounded instance might correspond to infeasible solutions in the original instance. On the other hand, when rounding up, some feasible solutions might become infeasible in the rounded instance.

Recently, new pseudo-polynomial algorithms have been proposed for Subset Sum (see Koiliaris and Xu [102] and Bringmann [25]). A natural idea is to use these to design an improved approximation scheme for Subset Sum. However, this seems to be difficult due to rounding issues discussed above. Shortly after this work was first published, Bringmann [26] explained this difficulty by giving a conditional lower bound on a quadratic approximation of Subset Sum.

### 4.1.1 Weak approximation for Subset Sum and application to Partition

Because of these rounding issues, it seems hard to design a general rounding scheme that, given a pseudo-polynomial algorithm for Subset Sum, produces an FPTAS for Subset Sum. What we can do, however, is to settle for a weaker notion of approximation.

**Definition 4.1.1** (Weak apx for Subset Sum)**.** *Let $Z^*$ be the optimal value for an instance $(Z, t)$ of Subset Sum. Given $(Z, t)$, a weak $(1-\varepsilon)$-approximation algorithm for Subset Sum returns $Z^H$ such that $(1-\varepsilon)Z^* \leq Z^H < (1+\varepsilon)t$.*

Compared to the traditional notion of approximation, here we allow a small violation of the packing constraint. This notion of approximation is interesting in itself. Indeed, it has been already considered in the stochastic regime for Knapsack [22].

Before going into details of constructing the weak $(1-\varepsilon)$-approximation algorithms for the Subset Sum, let us establish a relationship with the approximation for the Partition.

The key property of Partition here is its symmetric structure: If a subset $Z'$ violates the hard constraint $(t \leq \Sigma(Z') \leq (1+\varepsilon)t)$, then the set $Z - Z'$ is a good approximation and does not violate it (recall that in Partition problem we always have $t = \Sigma(Z)/2$).

**Observation 4.1.2.** *If we can weakly $(1-\varepsilon)$-approximate Subset Sum in time $\widetilde{\mathcal{O}}(T(n,\varepsilon))$, then we can $(1-\varepsilon)$-approximate Partition in the same $\widetilde{\mathcal{O}}(T(n,\varepsilon))$ time.*

*Proof.* Let $|Z| = n$ be the initial set of items. We run a weak $(1-\varepsilon)$-approximation algorithm for Subset Sum with target $b = \Sigma(Z)/2$. Let $Z^*$ denote the optimal partition of set $Z$:

$$Z^* = \operatorname*{arg\,max}_{Z' \subseteq Z,\, \Sigma(Z') \leq b} \Sigma(Z').$$

By the definition of weak $(1-\varepsilon)$-approximation for Subset Sum we get a solution $Z_W$ such that:

$$(1-\varepsilon)\Sigma(Z^*) \leq \Sigma(Z_W) \quad \text{and} \quad \Sigma(Z_W) < (1+\varepsilon)b$$

If $\Sigma(Z_W) \leq b$ then it is a correct solution for Partition. Otherwise we take a set $Z'_W = Z \setminus Z_W$. Because $\Sigma(Z)/2 = b$ we know that $\Sigma(Z'_W) < b$. Additionally we know, that $\Sigma(Z_W) < (1+\varepsilon)b$, so $(1-\varepsilon)b < \Sigma(Z'_W)$. Similarly, because $Z^* \leq b$, we have:

$$(1-\varepsilon)\Sigma(Z^*) \leq (1-\varepsilon)b < \Sigma(Z'_W) \leq \Sigma(Z^*) \leq b.$$

So $\Sigma(Z'_W)$ follows the definition of approximation for Partition. The running time follows because $T(n, 1/\varepsilon)$ must be superlinear (algorithm needs to read input at least) and we executed the weak $(1-\varepsilon)$-approximation Subset Sum algorithm only constant number of times. □

### 4.1.2 Constructing weak approximation algorithms for Subset Sum: a sketch

**Fact 4.1.3.** *Given an $\widetilde{\mathcal{O}}(T(n,t))$ exact algorithm for Subset Sum, we can construct a weak $(1-\varepsilon)$-approximation algorithm for Subset Sum working in time $\widetilde{\mathcal{O}}(T(n, \frac{n}{2\varepsilon}))$.*

*Proof.* We assume that the exact algorithm for the Subset Sum works also for multisets. We will address this issue in more detail in Section 4.2.1.

Let $Z = \{v_1, \ldots, v_n\}$ and $t$ constitute a Subset Sum instance. Let $I$ be the set of indices of elements of some optimal solution, and let OPT be their sum. Let us also introduce a scaled approximation parameter $\varepsilon' = \frac{\varepsilon}{4}$.

Let $k = \frac{2\varepsilon' t}{n}$. Define a rounded instance as follows: the (multi)-set of $\tilde{Z}$ contains a copy of $\tilde{v}_i = \left\lfloor \frac{v_i}{k} \right\rfloor$ for each $i \in \{1, \ldots, n\}$, and $\tilde{t} = \left\lfloor \frac{t}{k} \right\rfloor$.

Apply the exact algorithm $\mathcal{A}$ to the rounded instance $(\tilde{Z}, \tilde{t})$. Let $I'$ be the set of indices of elements of the solution found.

We claim that $\{v_i : i \in I'\}$ is a weak $(1-\varepsilon)$ approximation for $Z$ and $t$. First let us show that this solution is not much worse than OPT:

$$\begin{aligned}\sum_{i \in I'} v_i \geq k \sum_{i \in I'} \tilde{v}_i \geq k \sum_{i \in I} \tilde{v}_i = k \sum_{i \in I} \left\lfloor \frac{v_i}{k} \right\rfloor \geq \sum_{i \in I} (v_i - k) \\ \geq \mathrm{OPT} - nk = \mathrm{OPT} - 2\varepsilon' t \geq \mathrm{OPT}(1-\varepsilon).\end{aligned}$$

The last inequality holds because we can assume OPT $\geq t/2$ (see Section 4.2.3 for details).

Similarly, we can show that this solution does not violate the hard constraint by too much:

$$\sum_{i \in I'} v_i \leq \sum_{i \in I'} (k\tilde{v}_i + k) \leq nk + k \sum_{i \in I'} \tilde{v}_i \leq nk + \tilde{t}k \leq nk + k + t \leq 3\varepsilon' t + t \leq t(1+\varepsilon).$$

Finally, since the exact algorithm is applied to a (multi)-set of $n$ items with $\tilde{t} = \left\lfloor \frac{t}{k} \right\rfloor = \left\lfloor \frac{n}{2\varepsilon'} \right\rfloor$, the resulting algorithm runs in the claimed time. □

We state the above proof only to give the flavour of the basic form of reductions in this paper. Usually reductions that we will consider are more complex for technical reasons. One thing to note in particular is that the relation between $k$ and $\varepsilon$ is dictated by the fact, that there may be as many as $n$ items in the optimal solution. Given some control over the solution size, one can improve this reasoning (see Lemma 4.2.10).

### 4.1.3 Approximation via pseudo-polynomial time Subset Sum algorithm

Currently, the fastest pseudo-polynomial algorithm for Subset Sum runs in time $\widetilde{\mathcal{O}}(n+t)$, randomized. $\mathcal{S}(Z,t)$ denotes the set of all possible subsums of set $Z$ up to integer $t$.

**Theorem 4.1.4** (Bringmann [25])**.** *There is a randomized, one-sided error algorithm with running time* $\mathcal{O}(n+t\log t\log^3\frac{n}{\delta}\log n)$*, that returns a set* $Z' \subseteq \mathcal{S}(Z,t)$*, containing each element from* $\mathcal{S}(Z,t)$ *with probability at least* $1-\delta$*.*

This suffices to solve Subset Sum exactly with high probability. Here $\mathcal{S}(Z,t)$ is represented by a binary array which for a given index $i$ tells whether there is a subset that sums up to $i$. For our trade-off, we actually need a probabilistic guarantee on all elements of $\mathcal{S}(Z,t)$ simultaneously. Fortunately, this kind of bound holds for this algorithm as well (see Chapter 5 for detailed analysis).

**Corollary 4.1.5.** *There is a randomized* $\widetilde{\mathcal{O}}(n+t)$ *algorithm that computes* $\mathcal{S}(Z,t)$ *with a constant probability of success.*

The first case where this routine comes in useful occurs when all items are in the range $[\gamma t, t]$ (think of $\gamma$ as a trade-off parameter set to $\varepsilon^{-2/3}$). Note, that any solution summing to at most $t$ can consist of at most $1/\gamma$ such elements. This observation allows us to round the elements with lower precision and still maintain a good approximation ratio, as follows:

$$v_i' = \left\lfloor \frac{2v_i}{\gamma\varepsilon t} \right\rfloor, \qquad t' = \left\lfloor \frac{2t}{\gamma\varepsilon t} \right\rfloor = \left\lfloor \frac{2}{\gamma\varepsilon} \right\rfloor.$$

Bringmann's [25] algorithm on the rounded instance runs in time $\widetilde{\mathcal{O}}(n + t') = \widetilde{\mathcal{O}}(n + \frac{1}{\gamma\varepsilon})$ and returns an array of solutions with an additive error $\pm\varepsilon t$ with high probability (see Lemma 4.3.1). Similar reasoning about sparseness also applies if the number of items is bounded (i.e., when $n = \widetilde{\mathcal{O}}(\frac{\gamma}{\varepsilon})$). In that case Bringmann's [25] algorithm runs in time $\widetilde{\mathcal{O}}(\frac{\gamma}{\varepsilon^2})$ and provides the same guarantees (see Lemma 4.3.2 and also the next section).

### 4.1.4 Approximation via dense Subset Sum

Now we need a tool to efficiently solve the instances where all items are in range $[0, \gamma t)$, so-called dense instances. More formally, an instance consisting of $m$ items is dense if all items are in the range $[1, m^{\mathcal{O}(1)}]$. Intuitively, rounding does not work well for these instances since it introduces large rounding errors. On the other hand, if an instance contains many distinct numbers on a small interval, one can exploit its additive structure.

**Theorem 4.1.6** (Galil and Margalit [69])**.** *Let* $Z$ *be a set of* $m$ *distinct numbers in the interval* $(0, \ell]$ *such that*

$$m > 1000 \cdot \sqrt{\ell} \log \ell,$$

*and let* $L := \frac{100 \cdot \Sigma(Z) \ell \log \ell}{m^2}$.

*Then in* $\mathcal{O}(m + ((\ell/m) \log \ell)^2)$ *preprocessing time we can build a structure that can answer the following queries in constant time. In a query the structure receives a target number* $t \in (L, \Sigma(Z) - L)$ *and decides whether there is a* $Z' \subseteq Z$ *such that* $\Sigma(Z') = t$. *The structure is deterministic.*

In fact we will use a more involved theorem that can also construct a solution in $\mathcal{O}(\log(l))$ time but we omit it here to keep this section relatively free of technicalities (see Section 4.3.2 for a discussion regarding these issues).

Observe that $L = \widetilde{\mathcal{O}}(\ell^{1.5})$ (because $\Sigma(Z) < m\ell$) and the running time is bounded by $\widetilde{\mathcal{O}}(m + \ell)$ (because $\ell/m = \mathcal{O}(\sqrt{\ell})$). We will apply this result for the case $\ell = \gamma t$ (see Lemma 4.3.5). Recall, that Bringmann's [25] algorithm runs in time $\widetilde{\mathcal{O}}(m + t)$, which would be slower by the factor $\gamma$ (the trade-off parameter). For simplicity, within this overview we will assume, that Theorem 4.1.6 provides a data structure that can answer queries with the target numbers in $[0, \Sigma(Z)]$. In the actual proof, we need to overcome this obstacle, by merging this data structure with other structures, responsible for targets near the boundary, which we call *marginal* targets (see Lemma 4.3.3).

Suppose our instance consists of $m$ elements in the range $[0, \gamma t]$. We use the straightforward rounding scheme, as in the proof of Fact 4.1.3.

$$v_i' = \left\lfloor \frac{2mv_i}{\varepsilon t} \right\rfloor, \qquad t' = \left\lfloor \frac{2mt}{\varepsilon t} \right\rfloor = \left\lfloor \frac{2m}{\varepsilon} \right\rfloor.$$

We chose $\gamma t$ as the upper bound on item size, so that $\ell' = m\gamma/\varepsilon$ is an upper bound on $v_i'$. Now, if the number of items satisfies the inequality $\ell' < m^2$, then we can use the Theorem 4.1.6 with running time $\widetilde{\mathcal{O}}(m + \ell') = \widetilde{\mathcal{O}}(m + m\gamma/\varepsilon)$. This provides a data structure that can answer queries from the range that is of our interest (for a careful proof see Section 4.3).

Still, it can happen that most of the items are in the sparse instance (i.e., $\ell' \geq m^2$) and we cannot use the approach from [69]. In that case we use Theorem 4.1.4 again, with running time $\widetilde{\mathcal{O}}(m + \frac{\gamma}{\varepsilon^2})$ (see Lemma 4.3.2).

In the end, we are able to compute an array of solutions, for items in range $[0, \gamma t]$ in time $\widetilde{\mathcal{O}}(m + \frac{\gamma}{\varepsilon^2} + \frac{m\gamma^2}{\varepsilon^2})$ with additive error $\pm\varepsilon t$ and high probability (see Lemma 4.3.6). The last term in time complexity comes from handling the marginal queries.

### 4.1.5 A framework for efficient approximation

In this section we will sketch the components of our mechanism (see Algorithm 1). The mechanism combines pseudo-polynomial Bringmann's [25] algorithm with Galil and Margalit [69] algorithm for dense instances of Subset Sum.

**Algorithm 1** Roadmap for the weak $(1-\varepsilon)$-approximation for Subset Sum.
Input: item set $Z, t, \varepsilon$

1: ensure $OPT \geq t/2$
2: reduce $|Z|$ to $\widetilde{\mathcal{O}}(1/\varepsilon)$
3: **repeat**
4: partition items into $Z_{\text{large}}$ and $Z_{\text{small}}$
5: divide $[0, \gamma t]$ into $\ell = \mathcal{O}(\gamma \log(n)/\varepsilon) \cdot |Z_{\text{small}}|$ segments
6: round down small items
7: remove item repetitions in $Z_{\text{small}}$
8: **until** $\ell = \mathcal{O}(\gamma \log(n)/\varepsilon) \cdot |Z_{\text{small}}|$
9: build a data structure for large items
10: **if** $|Z_{\text{small}}| = \widetilde{\mathcal{O}}(\sqrt{\ell})$ **then**
11: build a data structure for small items
12: **else**
13: build data structures for marginals
14: exploit the density of the instance to cover the remaining case
15: **end if**
16: merge the data structures for large and small items

We begin by reducing the number of items in the instance $Z$ to roughly $\widetilde{\mathcal{O}}(1/\varepsilon)$ items to get a near linear running time (see Lemma 4.2.5). After that our goal is to divide items into *small* and *large* and process each part separately, as described earlier.

However, Theorem 4.1.6 requires a lower bound on the number of **distinct** items. To control this parameter, we merge identical items into larger ones, until each item appears at most twice. However, this changes the number of items, and so the procedure might have to be restarted. Lemma 4.2.7 guarantees that we require at most $\log n$ such refinement steps.

In the next phase we decide which method to use to solve the instance depending on its density (line 10). We encapsulate these methods into data structures (lines 11-14). Finally we will need to merge the solutions. For this task we introduce the concept of membership oracles (see Definition 4.2.8) that are based on FFT and backtracking to retrieve solutions (see Lemma 4.2.9). The simplified trade-off schema is presented on the Figure 4.1.

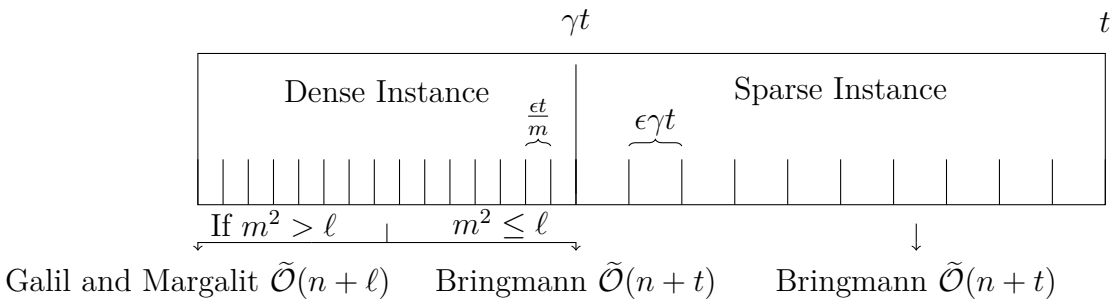



Figure 4.1: Overall schema of trade-off and usage of building blocks. The parameter $m$ denotes number of items in the dense instance, $n$ is the number of all elements, $\gamma$ is the trade-off parameter, $\ell$ is the upper bound on the item size after rounding, $t$ is the target sum. The buckets in the sparse/dense instance depict the rounding scheme for small and large items.

The final running time of our framework is $\widetilde{\mathcal{O}}(n + \frac{1}{\gamma\varepsilon} + \frac{\gamma}{\varepsilon^2} + \frac{\gamma^2}{\varepsilon^3})$ with high probability for any $\gamma(n, \varepsilon) > 0$ (see Lemma 4.3.9). For $\gamma = \varepsilon^{-2/3}$, this gives us an $\widetilde{\mathcal{O}}(n+\varepsilon^{-5/3})$ time weak $(1-\varepsilon)$-approximation approximation for Subset Sum.

## 4.2 Preprocessing

This section is devoted to simplifying the instance of Subset Sum to produce a more readable proof of the main algorithm. In here we deal with:

- multiplicities of the items,
- division of the instance into large and small items,
- proving that rounding preserves $\varepsilon$-closeness,
- reducing the number of items from $n$ to $\widetilde{\mathcal{O}}(1/\varepsilon)$ items.

The solutions to these problems are rather technical and well known in the community [25, 99, 100, 102]. We include it in here, because these properties are used in approximation algorithms [99, 100] and exact pseudo-polynomial algorithms [25, 102] communities separately. We expect that the reader may not be familiar with both of these technical toolboxes simultaneously and accompany this section with short historical references and pointers to the original versions of proofs.

We start with definitions regarding closeness of two solutions.

**Definition 4.2.1** ($(\varepsilon,t)$-closeness)**.** *We say that set $B$ is $(\varepsilon,t)$-close to $A$ if there is a surjection $\phi : A \to B$ such that $x - \varepsilon t \leq \phi(x) \leq x + \varepsilon t$. A Subset Sum instance $(Z_2,t)$ is $\varepsilon$-close to $(Z_1,t)$ if $\mathcal{S}(Z_2,t)$ is $(\varepsilon,t)$-close to $\mathcal{S}(Z_1,t)$.*

Sometimes, when there is no other notation on $t$, we use the notion of $\varepsilon$-closeness as a $(\varepsilon,t)$-close.

Usually the surjection from the definitions come by rounding down the item sizes and each item set get a moderately smaller total size. We also apply the notion of $(\varepsilon,t)$-closeness to binary arrays having in mind the sets they represent.

**Fact 4.2.2.** *If $A$ is $(\varepsilon,t)$-close to $\mathcal{S}(Z_1,t)$ and $B$ is $(\varepsilon,t)$-close to $\mathcal{S}(Z_2,t)$ then $A \oplus_t B$ is $(2\varepsilon,t)$-close to $\mathcal{S}(Z_1 \cup Z_2,t)$*

We also need to say, that there are no close elements in a set. It come in useful to show, that after rounding down all the elements are distinct.

**Definition 4.2.3** ($(x)$-distinctness)**.** *The set $S$ is said to be $(x)$-distinct if every interval of length $x$ contains at most one item from $S$. The set $S$ is said to be $(x,2)$-distinct if every interval of length $x$ contains at most two items from $S$.*

### 4.2.1 From multisets to sets

The general instance of Subset Sum may consist of plenty of items with equal size. Intuitively, these instances seem to be much simpler than instances where almost all items are different. The next lemma enables us to formally capture this intuition with the appropriate reduction. This lemma was proposed in [102, Lemma 2.2] but was also used in [25].

**Lemma 4.2.4** (cf. Lemma 2.2 from [102])**.** *Given a multiset $S$ of integers from $\{1,\ldots,t\}$, such that $|S| = n$ and the number of distinct items $||S||$ is $n'$, one can compute, in $\mathcal{O}(n \log n)$ time a multiset $T$, such that:*

- $\mathcal{S}(S,t) = \mathcal{S}(T,t)$
- $|T| \leq |S|$
- $|T| = \mathcal{O}(n' \log n)$
- *no element in $T$ has multiplicity exceeding two.*

*Proof.* We follow the proof from [102, Lemma 2.2], however the claimed bound on $|T|$ is only $\mathcal{O}(n' \log t)$ therein. Consider an element $x$ with the multiplicity $2k+1$. We can replace it with a single copy of $x$ and $k$ copies of $2x$ while keeping the multiset equivalent. If the multiplicity is $2k+2$ we need 2 copies of $x$ and $k$ copies of $2x$. We iterate over items from the smallest one and for each with at least 3 copies we perform the replacement as described above. Observe that this procedure generates only elements of form $2^i x$ where $i \leq \lceil \log n \rceil$ and $x$ is an element from $S$. This yields the bound on $|T|$. The routine can be implemented to take $\mathcal{O}(\log n)$ time for creating each new item using tree data structures. □

### 4.2.2 From $n$ items to $\widetilde{\mathcal{O}}(1/\varepsilon)$ items

To reduce number of items $n$ to $\widetilde{\mathcal{O}}(1/\varepsilon)$ Kellerer, Pferschy, and Speranza [100] gave a very intuitive construction that later found applications in Knapsack-type problems [99].

Intuitively, rounding scheme described in Section 4.1 could divide the items into $\mathcal{O}(n/\varepsilon)$ intervals and this would result with an $\varepsilon$-close instance to the original one. In here we start similarly but we want to get rid of factor $\mathcal{O}(n)$. We divide an instance to $k = \left\lceil \frac{1}{\varepsilon} \right\rceil$ intervals of length $\varepsilon t$, i.e., $I_j := (jt, (j+1)t]$. Next notice that for interval $I_j$ we do not need to store more than $\mathcal{O}(\left\lceil \frac{k}{j} \right\rceil)$ items, because their sum would exceed $t$ (this is the step where $\varepsilon$ factor will come in). Finally, the number of items is upper bounded (up to the constant factors):

$$\sum_{j=1}^{k} \left\lceil \frac{k}{j} \right\rceil \leq k \sum_{j=1}^{k} \frac{1}{j} < k \log k = \mathcal{O}\left( \frac{1}{\varepsilon} \log \frac{1}{\varepsilon} \right).$$

The last inequality is just an upper bound on harmonic numbers.

**Lemma 4.2.5.** *Given a Subset Sum instance $(Z, t)$, $|Z| = n$, one can find an $\varepsilon$-close instance $(Z_2, t)$ such that $|Z_2| = \mathcal{O}\Big(\frac{1}{\varepsilon} \log(\frac{n}{\varepsilon}) \log(n)\Big)$. The running time of this procedure is $\mathcal{O}(|Z| + |Z_2|)$.*

*Proof.* We begin with constructing $Z_1$ as follows. For $i = 1, \ldots, \log(\frac{2n}{\varepsilon})$ we round down each element in $Z \cap [\frac{t}{2^i}, \frac{t}{2^{i-1}})$ to the closest multiplicity of $\left\lfloor \frac{\varepsilon t}{2^{i+1}} \right\rfloor$. We neglect elements smaller than $\frac{\varepsilon t}{2n}$. Observe that $||Z_1|| = \mathcal{O}\Big(\frac{1}{\varepsilon} \log(\frac{n}{\varepsilon})\Big)$.

We argue that $(Z_1, t)$ is $\varepsilon$-close to $(Z, t)$. To see this, consider any subset $I \subseteq Z$ summing to at most $t$ and its counterpart $Y_1 \subseteq Z_1$. We lose at most $n \cdot \frac{\varepsilon t}{2n} = \frac{\varepsilon t}{2}$ by omitting items smaller than $\frac{\varepsilon t}{2n}$. Let $k_i = |I \cap [\frac{t}{2^i}, \frac{t}{2^{i-1}})|$ and $t_i$

denote the sum of elements in $I \cap [\frac{t}{2^i}, \frac{t}{2^{i-1}})$. Since each element in $[\frac{t}{2^i}, \frac{t}{2^{i-1}})$ has been decreased by at most $\frac{\varepsilon t}{2^{i+1}}$ and $k_i \cdot \frac{t}{2^i} \leq t_i$, we have

$$\Sigma(I) - \Sigma(Y_1) \leq \frac{\varepsilon t}{2} + \sum_{i=1}^{\log(\frac{2n}{\varepsilon})} k_i \cdot \frac{\varepsilon t}{2^{i+1}} \leq \frac{\varepsilon t}{2} + \sum_{i=1}^{\log(\frac{2n}{\varepsilon})} \frac{\varepsilon t_i}{2} \leq \varepsilon t.$$

In the end we take advantage of Lemma 4.2.4 to transform $Z_1$ into an equivalent multiset $Z_2$ such that $|Z_2| \leq ||Z_1|| \log(|Z_1|) = \mathcal{O}\Big(\frac{1}{\varepsilon} \log(\frac{n}{\varepsilon}) \log(n)\Big)$. □

Note, that we discarded items smaller than $\frac{\varepsilon t}{2n}$. We do this because the sum of these elements is just too small to influence the worst case approximation factor. We do not consider them just for the simplicity of analysis.

### 4.2.3 From one instance to small and large instances

First, we need a standard technical assumption, that says that we can cheaply transform an instance into one with a good bound on the solution. We need it just to simplify the proofs (e.g., it enables us to use Rounding Lemma 2.4.2 multiple times).

**Lemma 4.2.6.** *One may assume w.l.o.g. that for any Subset Sum instance* $OPT \geq \frac{t}{2}$.

*Proof.* Remove from the item set $Z$ all elements exceeding $t$ since they cannot belong to any solution. If $\Sigma(Z) \leq t$ then the optimal solution consists of all the items. Otherwise, consider a process in which $Y_1 = Z$ and in each turn, we obtain $Y_{k+1}$ by dividing $Y_k$ into two arbitrary non-empty parts and taking the one with a larger sum. We terminate the process when $Y_{last}$ contains only one item. Since $\Sigma(Y_1) > t$, $\Sigma(Y_{last}) \leq t$, and in each step the sum decreases by at most factor two, for some $k$ it must be $\Sigma(Y_k) \in [\frac{t}{2}, t]$. Because there is a feasible solution of value at least $\frac{t}{2}$, OPT cannot be lower. □

One of the standard techniques of solving Subset Sum is to separate the large and small items [99]. Usually, these approximations consider items greater and smaller than some trade-off parameter. Our techniques require a bound on the multiplicities of small items, which is provided by the next lemma.

**Lemma 4.2.7** (Partition into Small / Large Items)**.** *Given an instance* $(Z, t)$ *of Subset Sum, an approximation factor* $\varepsilon$*, and a trade-off parameter* $\gamma$*, one can deterministically transform the instance* $(Z, t)$*, in time* $\mathcal{O}(n \log^2 n)$*, to an* $\varepsilon$*-close instance* $(Z_{small} \cup Z_{large}, t)$ *such that:*

- $\forall z_s \in Z_{small}$, *it holds that* $z_s < \gamma t$,
- $\forall z_l \in Z_{large}$, *it holds that* $z_l \geq \gamma t$,
- *The set* $Z_{small}$ *is* $(\frac{\varepsilon t}{m \cdot \log n}, 2)$*-distinct where* $m = \mathcal{O}(|Z_{small}|)$*, i.e., after rounding there can be at most 2 occurrences of each item.*

*Proof.* We call an item $x$ *large* if $x \geq \gamma t$ and *small* otherwise. Let $Y_0$ be the initial set of small items and $m_0 = |Y_0|$, $q_0 = \text{pow}(\frac{\varepsilon t}{m_0 \log n})$. We round down the size of each small item to the closest multiplicity of $q_0$. Then we apply Lemma 4.2.4 to the set of small items to get rid of items with 3 or more copies. Note that this operation might introduce additional large items. We obtain a new set of small items $Y_1$ and repeat this procedure with notation $m_i = |Y_i|$, $q_i = \text{pow}(\frac{\varepsilon t}{m_i \log n})$. It holds that $m_{i+1} \leq m_i$ and $q_i \mid q_{i+1}$. We stop the process when $m_{i+1} \geq \frac{m_i}{2}$, which means there can be at most $\log n$ iterations. Let $m$ denote the final number of small items and $q \geq \frac{\varepsilon t}{4m \log n}$ – the last power of 2 used for rounding. All small items now occur with multiplicities at most 2.

Let us fix $Z_{\text{small}}$ as the set of small items after the modification above and likewise $Z_{\text{large}}$. In the $i$-th rounding step values of $m_i$ items are being decreased by at most $\frac{\varepsilon t}{m_i \log n}$, so each new instance is $\frac{\varepsilon}{\log n}$-close to the previous one. There are at most $\log n$ steps and the removal of copies keeps the instance equivalent, therefore $(Z_{\text{small}} \cup Z_{\text{large}}, t)$ is $\varepsilon$-close to $(Z, t)$. □

Our algorithm works independently on these two instances and produces two arrays $\varepsilon$-close to them. The construction below allows us to join these solutions efficiently. We want to use them even if we have only access to them by queries. We formalize this as an $(\varepsilon, t)$-membership-oracle. The asymmetry of the definition below will become clear in Lemma 4.2.10.

**Definition 4.2.8** ($(\varepsilon, t)$-membership-oracle)**.** *The* $(\varepsilon, t)$*-membership-oracle of a set* $X$ *is a data structure, that given an integer* $q$ *answers* ***yes/no*** *obeying following conditions:*

1. *if* $X$ *contains an element in* $[q - \varepsilon t, q + \varepsilon t]$*, then the answer is* ***yes****,*
2. *if the answer was* ***yes****, then* $X$ *contains an element in* $[q - 2\varepsilon t, q + 2\varepsilon t]$*.*

*A query to the oracle takes* $\widetilde{\mathcal{O}}(1)$ *time. Moreover, if the oracle answers* ***yes****, then it can return a witness* $x$ *in* $\widetilde{\mathcal{O}}(1)$ *time.*

Below we present an algorithm that can efficiently join solutions. We assume, that we have only $(\varepsilon, t)$-membership-oracle access to them and we want to produce an $(\varepsilon, t)$-membership-oracle of the merged solution.

**Lemma 4.2.9** (Merging solutions)**.** *Given $\mathcal{S}(Z_1, t)$ and $\mathcal{S}(Z_2, t)$ as $(\varepsilon, t)$-membership-oracles*

- *$S_1$ that is $(\varepsilon, t)$-close instance to $\mathcal{S}(Z_1, t)$,*
- *$S_2$ that is $(\varepsilon, t)$-close instance to $\mathcal{S}(Z_2, t)$,*

*we can, deterministically in $\widetilde{\mathcal{O}}(\frac{1}{\varepsilon})$ time, construct a $(2\varepsilon, t)$-membership-oracle for $\mathcal{S}(Z_1 \cup Z_2, t)$.*

*Proof.* For an ease of presentation, only in this proof we will use interval notation of inclusion, i.e., we will say that $(a, b] \sqcap A$ iff $\exists_x x \in (a, b] \wedge x \in A$. Let $p = \mathcal{O}(\varepsilon t)$. For each interval $(ip, (i+1)p]$ where $i \in \left\{0, \ldots \left\lfloor \frac{t}{p} \right\rfloor\right\}$ we query oracles whether $\mathcal{S}(Z_1, t)$ and $\mathcal{S}(Z_2, t)$ contain some element in the interval, having in mind that the answer is approximate. The number of intervals is $\mathcal{O}(\frac{1}{\varepsilon})$.

We store the answers in arrays $S_1$ and $S_2$, namely $S_j[i] = 1$ if the oracle for $\mathcal{S}(Z_j, t)$ answers yes for interval $(ip, (i+1)p]$.

$$S_1'[i] = \begin{cases} 1 & \text{if the oracle for } \quad (ip, (i+1)p] \sqcap S_1 \text{ or } i = 0 \\ 0 & \text{otherwise} \end{cases}$$

Then we perform a fast convolution on $S_1, S_2$ with FFT.

If $x \in \mathcal{S}(Z_1 \cup Z_2, t) \cap (kp, (k+1)p]$, then there is some $x_1 \in \mathcal{S}(Z_1, t)$ and $x_2 \in \mathcal{S}(Z_2, t)$ such that $x = x_1 + x_2$. We have $(S_1 \oplus_{\text{FFT}} S_2)[k] = \sum_{i=0}^{k} S_1[i] \cdot S_2[k-i]$ and thus $(S_1 \oplus_{\text{FFT}} S_2)[k']$ is nonzero for $k' = k$ or $k' = k+1$. This defines the rule for the new oracle. The additive error of the oracle gets doubled with the summation. On the other hand, if one of these fields is nonzero, then there are corresponding indices $i_1, i_2$ summing to $k$ or $k+1$. The second condition from Definition 4.2.8 allows the corresponding value $x_1$ to lie within one of the intervals with indices $i_1 - 1, i_1$, or $i_1 + 1$ and likewise for $x_2$. Therefore, the additive error is $\mathcal{O}(p) = \mathcal{O}(\varepsilon t)$.

Now, we promised only oracle output to our array. When a query comes, we scale down the query interval, then we check whether any of adjacent interval in our structure is nonzero (we lose a constant factor of $\mathcal{O}(\varepsilon)$ accuracy here) and output **yes** if we found it or **no** otherwise.

With additional polylogarithmic factors, we can also retrieve the solution. The idea is similar to backtracking from [100]. Namely, the fast convolution algorithm can compute the table of indexes of witnesses(of only one). We store an index of the witness if there is a solution or $-1$ otherwise. Then we ask the oracles of $\mathcal{S}(Z_1, t)$ and $\mathcal{S}(Z_2, t)$ for a solution with proper indexes and return the combination of those.

□

### 4.2.4 From exact solution to $\varepsilon$-close instance

In Section 4.1 we presented an overall approach of rounding elements and explained why it gives us the weak approximation of Subset Sum. Here we will focus on formally proving these claims.

In our subroutines, we round down the items, execute the exact algorithm on the rounded instance, and retrieve the solution. We want to argue, that in the end we lose only an additive factor of $\pm\varepsilon t$. We presented a sketch of this reasoning in Fact 4.1.3. For our purposes we will describe the procedure in the case, when the number of items in any solution is bounded by $k$ (i.e., we are interested only in $\mathcal{S}(Z,t)_k$). We can always assume $k \leq n$.

**Lemma 4.2.10.** *Given an exact algorithm that outputs the set $\mathcal{S}(Z,t)_k$ and works in time $T(n,t)$, where $n = |Z|$, we can construct an $(\varepsilon,t)$-membership-oracle of set $\mathcal{S}(Z,t)_k$ in time $\widetilde{\mathcal{O}}(n + T(n,k/\varepsilon))$.*

*If the exact algorithm retrieves solution in $\widetilde{\mathcal{O}}(1)$ time, then so does the oracle.*

*Proof.* For the sake of legibility, we assume that we are interested in $\mathcal{S}(Z,t)_{k-1}$. This allows us to write simpler formulas. Let $(z_i)$ denote the items. We perform rounding in the following way:

$$z'_i = \left\lfloor \frac{kz_i}{\varepsilon t} \right\rfloor, \qquad t' = \left\lfloor \frac{kt}{\varepsilon t} \right\rfloor = \left\lfloor \frac{k}{\varepsilon} \right\rfloor.$$

We run the exact algorithm on the rounded instance $(Z', t')$. It takes time $T(n,t') = \mathcal{O}(T(n,k/\varepsilon))$. This algorithm returns $\mathcal{S}(Z',t')_{k-1}$, which we store in array $Q[1,t']$. We construct $(\varepsilon,t)$-membership-oracle in array $Q'[1,t']$ as follows: we set $Q'[i] = 1$ iff $Q$ contains 1 in range $(i-2k, i+k]$. If we want to be to able retrieve a solution, we need to also remember a particular index $j(i) \in (i-2k, i+k]$ such that $Q[j(i)] = 1$. Such a data structure can be constructed in linear time with a queue. Given a query $q$, the oracle returns $Q'[q']$, where $q' = \left\lfloor \frac{kq}{\varepsilon t} \right\rfloor$. It remains to prove that Definition 4.2.8 is satisfied.

Let $I \subseteq Z$ be a set of at most $k-1$ items and $I'$ be the set of their counterparts after rounding. Since for all $z_i \in Z$ it holds

$$\frac{kz_i}{\varepsilon t} - 1 \leq z'_i \leq \frac{kz_i}{\varepsilon t},$$

we obtain

$$\frac{k \cdot \Sigma(I)}{\varepsilon t} - k + 1 \leq \Sigma(I') \leq \frac{k \cdot \Sigma(I')}{\varepsilon t}. \tag{4.1}$$

Therefore, if $\Sigma(I) \in [q - \varepsilon t, q + \varepsilon t]$, then

$$\frac{kq}{\varepsilon t} - 2k + 1 = \frac{k \cdot (q - \varepsilon t)}{\varepsilon t} - k + 1 \leq \Sigma(I'),$$
$$\Sigma(I') \leq \frac{k \cdot (q + \varepsilon t)}{\varepsilon t} = \frac{kq}{\varepsilon t} + k,$$

and $\Sigma(I') \in (q' - 2k, q' + k]$, because $\Sigma(I')$ is integer. On the other hand, we can invert relation (4.1) to obtain

$$\frac{\varepsilon t}{k} \cdot \Sigma(I') \leq \Sigma(I) \leq \frac{\varepsilon t}{k} \cdot (\Sigma(I') + k - 1)\,.$$

To satisfy the second condition we assume $\Sigma(I') \in (q' - 2k, q' + k]$ and check that

$$q - 2\varepsilon t = \frac{\varepsilon t}{k} \cdot \left(\frac{kq}{\varepsilon t} - 2k\right) \leq \frac{\varepsilon t}{k} \cdot (q' - 2k + 1) \leq \Sigma(I),$$
$$\Sigma(I) \leq \frac{\varepsilon t}{k} \cdot (q' + 2k - 1) \leq q + 2\varepsilon t,$$

what finishes the proof. □

First let us restate Lemma 2.4.2 from Chapter 2.

**Lemma 2.4.2.** *For $k$ natural numbers $x_1, x_2, \ldots, x_k$ and positive $q, \varepsilon$ such that $q \leq \sum_{i=1}^{k} x_i$ and $0 < \varepsilon < 1$, it holds:*

$$(1 - \varepsilon) \sum_{i=1}^{k} x_i < \frac{q\varepsilon}{k} \sum_{i=1}^{k} \left\lfloor \frac{kx_i}{q\varepsilon} \right\rfloor \leq \sum_{i=1}^{k} x_i.$$

We apply Lemma 2.4.2 with $\{z_1, \ldots, z_k\} = Y$, $q = t/2$ and $k$ and $\varepsilon$ as in the statement. It guarantees that:

$$(1 - \varepsilon)\Sigma(Y) \leq \frac{\varepsilon t}{2k}\Sigma(Y').$$

And finally, $(1 - \varepsilon)\Sigma(Y) \geq \Sigma(Y) - \varepsilon t$ (because we are only interested in solutions smaller than $t$). So if $Y$ is an optimal solution, then an exact algorithm after rounding returns candidate solution greater or equal $\Sigma(Y) - \varepsilon t$.

Conversely, it can turn out that an exact algorithm finds candidate solution with a sum greater than $q$ (this is where we can violate the hard constraint). We need to bound it as well (because the definition of $(\varepsilon, t)$-membership-oracle requires that). Note, that analogous argument proves it.

Namely, the solution can consist of at most $k$ items and each of them lose only $\mathcal{O}(\varepsilon t/k)$. Moreover, exact oracle gave us only the solution that its rounded version sums up to exactly $t'$. Formally, we prove it again with Lemma 2.4.2 with the same parameters as before. By dividing both sides by $(1-\varepsilon)$ we know that:

$$\sum_{i=1}^{k} \left\lfloor \frac{2kz_i}{t\varepsilon} \right\rfloor = \lfloor k \rfloor \cdot \varepsilon.$$

Once again, we can use Lemma 2.4.2 with the same parameters (we divided both sides by $(1-\varepsilon) > 0$):

$$\sum_{i=1}^{k} x_i \leq \left(\frac{1}{1-\varepsilon}\right) \frac{\varepsilon t}{2k} \sum_{i=1}^{k} \left\lfloor \frac{2kz_i}{t\varepsilon} \right\rfloor.$$

The right side satisfies:

$$\left(\frac{1}{1-\varepsilon}\right) \frac{\varepsilon t}{2k} \sum_{i=1}^{k} \left\lfloor \frac{2kz_i}{t\varepsilon} \right\rfloor = \left(\frac{1}{1-\varepsilon}\right) \frac{\varepsilon t}{2k} \left\lfloor \frac{2k}{\varepsilon} \right\rfloor \leq \frac{1}{1-\varepsilon} t < (1+2\varepsilon)t.$$

The constant before $\varepsilon$ does not change much since we only need $(\mathcal{O}(\varepsilon), t)$-membership-oracle (we can always rescale the approximation factor by setting $\varepsilon' = \varepsilon/2$ in the beginning).

The main obstacle with returning a solution that obeys the capacity constraint comes from the above lemma. If we could provide a reduction from an exact algorithm without widening the interval $[q-\varepsilon t, q+\varepsilon t]$, this would automatically entail a strong approximation for Subset Sum. This seems unlikely due to conditional hardness result for a strong subquadratic approximation for Subset Sum [26].

In the end, we need to prove, that an $(\varepsilon, t)$-membership-oracle gives us the correct solution for weak $(1-\varepsilon)$-approximation Subset Sum.

**Lemma 4.2.11.** *Given an $(\varepsilon, t)$-membership-oracle of $\mathcal{S}(Z, t)$, we can read the answer to the weak $(1-\mathcal{O}(\varepsilon))$ approximation for Subset Sum in time $\widetilde{\mathcal{O}}(\frac{1}{\varepsilon})$.*

*Proof.* We query the oracle for $q = i \cdot \varepsilon t$ for $i = 0, \ldots, \frac{1}{\varepsilon}$. Each query takes time $\widetilde{\mathcal{O}}(1)$ and if the interval $[q-\varepsilon t, q+\varepsilon t]$ contains an $x \in \mathcal{S}(Z, t)$, then the oracle returns an element within $[x-\mathcal{O}(\varepsilon t), x+\mathcal{O}(\varepsilon t)]$. If $\mathrm{OPT} < (1-\mathcal{O}(\varepsilon))t$, then the oracle will return a witness within $(\mathrm{OPT}-\mathcal{O}(\varepsilon t), \mathrm{OPT}]$. Otherwise the witness might belong to $(t, (1+\mathcal{O}(\varepsilon))t]$.

By taking advantage of Lemma 4.2.6, we can assume that $\mathrm{OPT} \geq t/2$, therefore the relative error gets bounded with respect to OPT. □

## 4.3 The weak $(1-\varepsilon)$-approximation algorithm for Subset Sum

### 4.3.1 Large items

We plan to use Theorem 4.1.4 to compute $\mathcal{S}(Z_{\text{large}}, t)$ on a large instance. On that instance, this algorithm is more efficient than Kellerer et al. [100] algorithm because one can round items less aggressively.

**Lemma 4.3.1** (Algorithm for Large Items)**.** *Given a large instance $(Z_{large}, t)$ of Subset Sum (i.e., all items are greater than $\gamma t$), we can construct an $(\varepsilon, t)$-membership-oracle of $\mathcal{S}(Z_{large}, t)$ in randomized $\widetilde{\mathcal{O}}(n + \frac{1}{\gamma\varepsilon})$ time with a constant probability of success.*

*Proof.* We use Bringmann's [25] algorithm, namely Corollary 4.1.5, that solves the Subset Sum problem exactly. Since all elements are greater than $\gamma t$, any subset that sums up to at most $t$ must contain at most $\frac{1}{\gamma}$ items. The parameter $k$ in Lemma 4.2.10 is an upper bound on number of elements in the solution, hence we set $k = \frac{1}{\gamma}$. The Bringmann's [25] algorithm runs in time $\widetilde{\mathcal{O}}(n + t)$ and Lemma 4.2.10 guarantees that we can build an $(\varepsilon, t)$-membership-oracle in time $\widetilde{\mathcal{O}}(n + k/\varepsilon) = \widetilde{\mathcal{O}}(n + 1/(\gamma\varepsilon))$, which is what we needed.

□

### 4.3.2 Small items

Now we need an algorithm that solves the problem for small items. As mentioned in Section 4.1 we consider two cases depending on the density of instance. The initial Subset Sum instance consists of $n$ elements. The $m$ is the number of elements in the small instance and let $m' = \mathcal{O}(m \log n)$ be as in Lemma 4.2.7. For now, we assume that the set of elements is $(\varepsilon t/m')$-distinct (we deal with multiplicities 2 in Lemma 4.3.7).

Let $q = \varepsilon t/m'$ be the rounding parameter (the value by which we divide) and $\ell = \gamma m'/\varepsilon = \mathcal{O}(\frac{\gamma m \log n}{\varepsilon})$ be the upper bound on items sizes in the small instance after rounding. Parameter $L = \mathcal{O}(\Sigma(S) \cdot \frac{l}{m^2})$ describes the boundaries of Theorem 4.1.6. We deliberately use $\mathcal{O}$ notation to hide constant factors (note that Galil and Margalit [69] algorithm requires that $m > 1000 \cdot \sqrt{l} \log l$).

**Lemma 4.3.2** (Small items and $m^2 < \ell \log^2 \ell$)**.** *Suppose we are given an instance $(Z_{small}, t)$ of Subset Sum (i.e., all items are smaller than $\gamma t$) with*

*size satisfying $m^2 < \ell \log^2 \ell$. Then we can compute $(\varepsilon, t)$-membership-oracle of $\mathcal{S}(Z_{small}, t)$ in randomized $\widetilde{\mathcal{O}}(m + \frac{\gamma}{\varepsilon^2})$ time.*

*Proof.* In here we need to deal with the case, where *small* instance is sparse. So just as in the proof of Lemma 4.3.1, we can use Bringmann's [25] algorithm.

We use the reduction from exact to weak $(1-\varepsilon)$-approximation algorithm for Subset Sum from Lemma 4.2.10. We set $m$ as the maximal number of items in the solution, as there are at most $m$ small items. Recall that $\ell$ is $\widetilde{\mathcal{O}}(m\gamma/\varepsilon)$. This gives us $m^2 = \widetilde{\mathcal{O}}(\ell) = \widetilde{\mathcal{O}}(\frac{m\gamma}{\varepsilon})$. After dividing both sides by $m$ we obtain $m = \widetilde{\mathcal{O}}(\frac{\gamma}{\varepsilon})$.

Combining Corollary 4.1.5 and Lemma 4.2.10 allows us to construct an $(\varepsilon, t)$-membership-oracle in $\widetilde{\mathcal{O}}(m + T(m, m/\varepsilon)) = \widetilde{\mathcal{O}}(m + \frac{\gamma}{\varepsilon^2})$ randomized time. □

Finally, we have to handle the harder $m^2 \geq \ell \log^2 \ell$ case. In this situation we again consider two cases. The Galil and Margalit [69] algorithm allows only to ask queries in the range $(L,\ \Sigma(S) - L)$ where $L = \mathcal{O}(\Sigma(S) \cdot \frac{l}{m^2})$. In the next lemma we take care of ranges $[0, L]$ and $[\Sigma(S) - L, \Sigma(S)]$. We focus on the range $[0, L]$, because the sums within $[\Sigma(S) - L, \Sigma(S)]$ are symmetric to $[0, L]$.

**Lemma 4.3.3** (Small items, range $(0, L)$)**.** *Given an instance $(Z_{small}, t)$ of Subset Sum, such that $|Z_{small}| = m$ and the items' sizes are at most $\gamma t$, we can compute an $(\varepsilon, t)$-membership-oracle for $\mathcal{S}(Z_{small}, L)$ in time $\widetilde{\mathcal{O}}(m + \frac{m\gamma^2}{\varepsilon^2})$.*

*Proof.* We round down items with rounding parameter $q = \varepsilon t/m' = \Omega(\frac{\varepsilon t}{m \log n})$ and denote the set of rounded items as $Z'_{\text{small}}$. After scaling down we have $L' = \Sigma(Z'_{\text{small}}) \cdot \frac{c\ell}{m^2}$ (note that we only replace $\Sigma(Z_{\text{small}})$ with $\Sigma(Z'_{\text{small}})$ and $\ell$ remains the same). Recall that $\ell = \mathcal{O}(\frac{m\gamma \log n}{\varepsilon})$.

The total sum of items in $Z'_{\text{small}}$ is smaller or equal to $\ell m$ (because there are $m$ elements of size at most $\ell$). Hence ,$L' = \mathcal{O}(\ell^2/m) = \mathcal{O}(\frac{\gamma^2 m \log^2 n}{\varepsilon^2})$. Therefore Bringmann's [25] algorithm runs in time $\widetilde{\mathcal{O}}(m + L') = \widetilde{\mathcal{O}}(m + \frac{m\gamma^2}{\varepsilon^2})$. Combining it with the analysis of the Lemma 4.2.10 gives us an $(\varepsilon, t)$-membership-oracle for $\mathcal{S}(Z_{\text{small}}, L)$. □

### 4.3.3 Applying additive combinatorics

Before we proceed forward, we need to present the full theorem of Galil and Margalit [68, Theorem 6.1] (in Section 4.1.4 we presented only a short version to keep it free from technicalities). We need a full running time complexity

(with dependence on $\ell, m, \Sigma(S)$). We stated it in here with a slight change of notation (e.g., [68] use $S_A$, however we use notation $\Sigma(A)$ from [102]).

**Theorem 4.3.4** (Theorem 6.1 from [68]). *Let $A$ be a set of $m$ different numbers in interval $(0, \ell]$ such that*

$$m > 1000 \cdot \ell^{0.5} \log_2 \ell;$$

*then we can build in $\mathcal{O}\left(m + ((\ell/m)\log l)^2 + \frac{\Sigma(A)}{m^2}\ell^{0.5}\log^2 \ell\right)$ preprocessing time a structure which allows us to solve the Subset Sum problem for any given integer $N$ in the interval $(L, \Sigma(A) - L)$. Solving means finding a subset $B \subseteq A$, such that $\Sigma(B) \leq N$ and there is no subset $C \subseteq A$ such that $\Sigma(B) < \Sigma(C) \leq N$. An optimal subset $B$ is build in $\mathcal{O}(\log \ell)$ time per target number and is listed in time $\mathcal{O}(|B|)$. For finding the optimal sum $\Sigma(B)$ only, the preprocessing time is $\mathcal{O}\left(m + ((\ell/m)\log \ell)^2\right)$ and only constant time is needed per target number.*

Galil and Margalit [68] defined $L := \frac{100 \cdot \Sigma(A)\ell^{0.5}\log_2 \ell}{m}$, however in different version, [69] improved it to $L := \mathcal{O}(\Sigma(A)\frac{\ell}{m^2})$ without any change in the running time [70]. We obtain a subquadratic algorithm for both of these possible choices of $L$. We use the improved version [69] because it guarantees a better running time.

**Lemma 4.3.5** (Small items, range $(L, \Sigma(S) - L)$). *Given a small instance $(Z_{small}, t)$ of Subset Sum (i.e., all items are smaller than $\gamma t$) such that $Z_{small}$ is $(\varepsilon t/m')$-distinct (where $m' = \mathcal{O}(m \log n)$), we can compute an $(\varepsilon, t)$-membership-oracle of $\mathcal{S}(Z_{small}, t) \cap (L, \Sigma(Z_{small}) - L)$ in time $\widetilde{\mathcal{O}}(n + \left(\frac{\gamma}{\varepsilon}\right)^2 + \frac{\gamma}{\varepsilon} \cdot \left(\frac{\gamma n}{\varepsilon}\right)^{0.5})$.*

*Proof.* We round items to multiplicities of $q = \varepsilon t/m'$. Precisely:

$$z'_i = \left\lfloor \frac{z_i}{q} \right\rfloor, \qquad t' = \left\lfloor \frac{t}{q} \right\rfloor = \left\lfloor \frac{m'}{\varepsilon} \right\rfloor.$$

We know that $z_i < \gamma t$. Therefore

$$z'_i \leq \frac{z_i}{q} < \frac{\gamma t}{q} = \frac{\gamma m'}{\varepsilon} = \ell.$$

By the same inequalities as in the proof of Lemma 4.2.10 we know that if we compute $\mathcal{S}(Z'_{\text{small}}, t')$ and multiply all results by $q$, we obtain an $(\varepsilon, t)$-membership-oracle for $\mathcal{S}(Z_{\text{small}}, t)$.

**Checking conditions of the algorithm** Now, we check that we satisfy all assumptions of Galil and Margalit [69] algorithm on the rounded instance $Z'_{\text{small}}$. First, note that $m^2 < \ell \log^2 \ell$, $\ell$ is the upper bound on the items' sizes in $Z'_{\text{small}}$, and we know that all items in $Z'_{\text{small}}$ are distinct because we assumed that $Z_{\text{small}}$ is $(\varepsilon t/m')$-distinct.

**Preprocessing** Next Galil and Margalit [69] algorithm constructs a data structure on the set of rounded items $Z'_{\text{small}}$. The preprocessing of Galil and Margalit [69] algorithm requires

$$\mathcal{O}\left(m + (\ell/m \log \ell)^2 + \frac{\Sigma(Z'_{\text{small}})}{m^2} \ell^{0.5} \log^2 \ell\right)$$

time. If we put it in terms of $m, \varepsilon, t$ and hide polylogarithmic factors we see that preprocessing runs in:

$$\widetilde{\mathcal{O}}\left(m + \left(\frac{\gamma}{\varepsilon}\right)^2 + \frac{\gamma}{\varepsilon}\left(\frac{\gamma m}{\varepsilon}\right)^{0.5}\right)$$

because $\Sigma(Z'_{\text{small}}) \leq \ell m$.

**Queries** With this data structure we need to compute a set $\varepsilon$-close to $\mathcal{S}(Z_{\text{small}}, t) \cap (L,\, \Sigma(Z'_{\text{small}}) - L)$. After scaling down we have:

$$L' = \widetilde{\mathcal{O}}\left(\Sigma(Z'_{\text{small}}) \cdot \frac{\ell}{m^2}\right) = \widetilde{\mathcal{O}}(\frac{\ell^2}{m}) = \widetilde{\mathcal{O}}(\frac{\gamma^2 m^2}{m\varepsilon^2}) = \widetilde{\mathcal{O}}(\frac{m\gamma^2}{\varepsilon^2}).$$

Naively, one could run queries for all elements in a range $(L', \Sigma(Z'_{\text{small}}) - L')$ and check if there is a subset of $Z'_{\text{small}}$ that sums up to the query value. However, this is too expensive. In order to deal with this issue, we take advantage of the fact that each query returns the closest set whose sum is smaller or equal to the query value.

Since we have rounded down items with $q = \frac{\varepsilon t}{m'}$, we only need to ask $\frac{\varepsilon t}{q} = \widetilde{\mathcal{O}}(m)$ queries in order to learn sufficient information. The queries reveal if $Z_{\text{small}}$ contains at least one element in each range $[i\varepsilon t, (i+1)\varepsilon t)$, what matches the definition of the $(\varepsilon, t)$-membership-oracle.

**Retrieving the solution** Galil and Margalit [69] algorithm can retrieve the solution in time $\mathcal{O}(\log \ell)$.

This finalizes the construction of the $(\varepsilon, t)$-membership-oracle. The running time is dominated by the preprocessing time.

□

### 4.3.4 Combining the algorithms

Here, we combine the algorithms for small items.

**Lemma 4.3.6** (Small Items)**.** *Given a $(Z_{small}, t)$ instance of Subset Sum (i.e., all elements in $Z_{small}$ are smaller than $\gamma t$), such that the set $Z_{small}$ is $(\varepsilon t/m)$-distinct, we can compute an $(\varepsilon, t)$-membership-oracle of $\mathcal{S}(Z_{small}, t)$ in time $\widetilde{\mathcal{O}}(m + \frac{\gamma}{\varepsilon^2} + \frac{m\gamma^2}{\varepsilon^2})$ with high probability.*

*Proof.* We combine two cases:

**Case When $m^2 < \ell \log^2 \ell$:** use algorithm from Lemma 4.3.2 that runs in $\widetilde{\mathcal{O}}(m + \frac{\gamma}{\varepsilon^2})$ time.

**Case When $m^2 \geq \ell \log^2 \ell$:** Lemma 4.3.5 returns an $(\varepsilon, t)$-membership-oracle that answers queries within set $\mathcal{S}(Z_{\text{small}}, t) \cap (L, \Sigma(Z_{\text{small}}) - L)$. It requires $\widetilde{\mathcal{O}}(nm + \left(\frac{\gamma}{\varepsilon}\right)^2 + \frac{\gamma}{\varepsilon} \cdot \left(\frac{\gamma m}{\varepsilon}\right)^{0.5})$ time.

We combine it (using Lemma 4.2.9) with the $(\varepsilon, t)$-membership-oracle that gives us answers to a set $\mathcal{S}(Z_{\text{small}}, t) \cap [0, L]$ from Lemma 4.3.3. This oracle can be constructed in time $\widetilde{\mathcal{O}}(m + \frac{m\gamma^2}{\varepsilon^2})$. The oracle for interval set $[\Sigma(Z_{\text{small}}) - L, \Sigma(Z_{\text{small}})]$ is obtained by symmetry.

**Running Time:** The running time of merging the solutions from Lemma 4.2.9 is $\widetilde{\mathcal{O}}(1/\varepsilon)$ which is suppressed by the running time of Lemma 4.3.3 and Lemma 4.3.5. Factor $\widetilde{\mathcal{O}}(\left(\frac{\gamma}{\varepsilon}\right)^2)$ is suppressed by $\widetilde{\mathcal{O}}\left(\left(\frac{m\gamma^2}{\varepsilon^2}\right)\right)$.

Term $\frac{\gamma}{\varepsilon} \cdot \left(\frac{\gamma m}{\varepsilon}\right)^{0.5}$ is also suppressed by $\widetilde{\mathcal{O}}(\frac{m\gamma^2}{\varepsilon^2})$. The randomization comes from Lemma 4.3.3. □

The Lemma 4.2.7 allowed us to partition our instance into small and large items. We additionally know that each interval of length $\varepsilon t/m'$ contains at most 2 items. However in the previous proofs we assumed there can be only one such item, i.e., the set should be $(\varepsilon t/m')$-distinct.

**Lemma 4.3.7** (From multiple to distinct items)**.** *Given an instance $(Z_{small}, t)$ of Subset Sum, where $|Z_{small}| = m$ and zzsmall is $(\varepsilon t/m', 2)$-distinct for $m' = \mathcal{O}(m \log n)$, we can compute an $(\varepsilon, t)$-membership-oracle for instance $(Z_{small}, t)$ in $\widetilde{\mathcal{O}}(n + \frac{\gamma}{\varepsilon^2} + \frac{n\gamma^2}{\varepsilon^2})$ time with high probability.*

*Proof.* We divide the set $Z_{\text{small}}$ into two sets $Z^1_{\text{small}}$ and $Z^2_{\text{small}}$ such that $Z_{\text{small}} = Z^1_{\text{small}} \cup Z^2_{\text{small}}$, the sets $Z^1_{\text{small}}$,$Z^2_{\text{small}}$ are disjoint, $(\varepsilon t/m')$-distinct,

and have size $\Omega(m)$. This can be done by sorting $Z_{\text{small}}$ and dividing items into odd-indexed and even-indexed . It takes $\widetilde{\mathcal{O}}(m)$ time.

Next we use Lemma 4.3.6 to compute an $(\varepsilon, t)$-membership-oracle for $(Z^1_{\text{small}}, t)$ and $(Z^2_{\text{small}}, t)$, and merge them using Lemma 4.2.9. □

Now, we combine the solution for small with solution for large items.

**Theorem 4.3.8.** *Let $0 < \gamma$ be a trade-off parameter (that depends on $n, \varepsilon$). Given an $(Z, t)$ instance of Subset Sum, we can construct the $(\varepsilon, t)$-membership-oracle of instance $\mathcal{S}(Z, t)$ in $\widetilde{\mathcal{O}}(n + \frac{1}{\gamma\varepsilon} + \frac{\gamma}{\varepsilon^2} + \frac{n\gamma^2}{\varepsilon^2})$ time with high probability.*

*Proof.* We start with Lemma 4.2.7, that in $\mathcal{O}(n \log^2 n)$ time partitions the set into $Z_{\text{large}}$ and $Z_{\text{small}}$, such that $Z_{\text{small}}$ is $(\frac{\varepsilon t}{m \log n}, 2)$-distinct, where $m = |Z_{\text{small}}|$. To deal with small items, we use Lemma 4.3.7. The algorithm for small items returns an $(\varepsilon, t)$-membership-oracle of $\mathcal{S}(Z_{\text{small}}, t)$. For large items we can use Lemma 4.3.1. It also returns an $(\varepsilon, t)$-membership-oracle of $\mathcal{S}(Z_{\text{large}}, t)$.

Finally, we use Lemma 4.2.9 to merge these oracles in time $\widetilde{\mathcal{O}}(1/\varepsilon)$. All the subroutines run with a constant probability of success. □

Finally, we have combined all the pieces and we can get a faster algorithm for weak $(1 - \varepsilon)$-approximation for Subset Sum.

**Corollary 4.3.9** (Subset Sum with tradeoff)**.** *There is a randomized weak $(1-\varepsilon)$-approximation algorithm for Subset Sum running in $\widetilde{\mathcal{O}}(n+\frac{1}{\gamma\varepsilon}+\frac{\gamma}{\varepsilon^2}+\frac{n\gamma^2}{\varepsilon^2})$ time with high probability for any $\gamma(n, \varepsilon) > 0$.*

*Proof.* It follows from Lemma 4.2.11 and Theorem 4.3.8. □

The weak $(1 - \varepsilon)$-approximation Subset Sum guarantess the approximation for Partition via Corollary 4.1.2.

**Corollary 4.3.10** (Partition with trade-off)**.** *There is a randomized $(1-\varepsilon)$-approximation algorithm for Partition running in $\widetilde{\mathcal{O}}(n + \frac{1}{\gamma\varepsilon} + \frac{\gamma}{\varepsilon^2} + \frac{n\gamma^2}{\varepsilon^2})$ time with high probability for any $\gamma(n, \varepsilon) > 0$.*

To get running time of form $\widetilde{\mathcal{O}}(n + 1/\varepsilon^c)$ and prove our main result we need to reduce the number of items from $n$ to $\widetilde{\mathcal{O}}(1/\varepsilon)$ and choose the optimal $\gamma$.

**Theorem 4.3.11** (Weak approximation for Subset Sum)**.** *There is a randomized weak $(1 - \varepsilon)$-approximation algorithm for Subset Sum running in $\widetilde{\mathcal{O}}\left(n + \varepsilon^{-\frac{5}{3}}\right)$ time.*

*Proof.* We apply Lemma 4.2.5 to reduce the number of items to $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon}\right)$ and guarantee that the instance is $\mathcal{O}(\varepsilon)$-close. Then we use Corollary 4.3.9 with $\gamma = \varepsilon^{\frac{2}{3}}$. □

Finally, for Partition we get that:

**Theorem 4.3.12** (Approximation for Partition)**.** *There is a randomized* $(1-\varepsilon)$*-approximation algorithm for Partition running in* $\widetilde{\mathcal{O}}\left(n + \varepsilon^{-\frac{5}{3}}\right)$ *time.*

*Proof.* We use Corollary 4.1.2 that reduces approximating Partition to weak $(1-\varepsilon)$-approximation of Subset Sum, and Theorem 4.3.11 to solve it efficiently. □

# Part III

# Equivalences in the Tropical Semirings

# Chapter 5

# On problems equivalent to (min,+)-convolution

In recent years, significant progress has been made in explaining the apparent hardness of improving upon the naive solutions for many fundamental polynomially solvable problems. This progress has come in the form of conditional lower bounds – reductions from a problem assumed to be hard. The hard problems include 3SUM, All-Pairs Shortest Path, SAT, Orthogonal Vectors, and others.

In the $(\min, +)$-convolution problem, the goal is to compute a sequence $(c[i])_{i=0}^{n-1}$, where $c[k] = \min_{i=0,\ldots,k} \{a[i] + b[k-i]\}$, given sequences $(a[i])_{i=0}^{n-1}$ and $(b[i])_{i=0}^{n-1}$. This can easily be done in $\mathcal{O}(n^2)$ time, but no $\mathcal{O}(n^{2-\varepsilon})$ algorithm is known for $\varepsilon > 0$. We undertake a systematic study of the $(\min, +)$-convolution problem as a hardness assumption.

First, we establish the equivalence of this problem to a group of other problems, including variants of the classic knapsack problem and problems related to subadditive sequences. The $(\min, +)$-convolution problem has been used as a building block in algorithms for many problems, notably problems in stringology. It has also appeared as an ad hoc hardness assumption. Second, we investigate some of these connections and provide new reductions and other results. We also explain why replacing this assumption with the SETH might not be possible for some problems.

We also prove several related results. Notably, we improve approximation schemes for 3SUM, Min-Plus Convolution, and Tree Sparsity. Finally, we argue why breaking the quadratic barrier for approximate Knapsack is

The picture illustrates a *convoluted* ball of wool. It is an allusion to the name *Min-Plus Convolution* and its applications in stringology.

unlikely by giving an $\Omega((n+1/\varepsilon)^{2-o(1)})$ conditional lower bound.

**Outline:** Theorem 1.3.11 is split into five implications, presented separately as Theorems 5.1.1, 5.1.3, 5.1.4, 5.1.5 and 5.2.5. While Theorem 1.3.11 has a relatively short and simple statement, it is not the strongest possible version of the equivalence. In particular, one can show analogous implications for subpolynomial improvements, such as the $\mathcal{O}(n^2/2^{\Omega(\log n)^{1/2}})$ algorithm for Min-Plus Convolution presented by Williams [157]. The theorems listed above contain stronger versions of the implications. The proof of Theorem 5.1.5 has been independently given in [18]. We present it here because it is the first step in the ring of reductions and introduces the essential technique of Vassilevska and Williams [154].

Section 5.3 is devoted to the remaining arrows in Figure 1.2. In Subsection 5.3.1, we show that by using Theorem 1.3.11, we can obtain an alternative proof of the equivalence of MCSP and Max-Plus Convolution (and thus also Min-Plus Convolution), which is much simpler than that presented in [104]. In Subsection 5.3.2, we show that Tree Sparsity reduces to Max-Plus Convolution, complementing the opposite reduction shown in [18]. We also provide some observations on the possible equivalence between $l_\infty$-Necklace Alignment and Max-Plus Convolution in Subsection 5.3.3.

The relation between Max-Plus Convolution and 3SUM implies that we should not expect the new conjecture to follow from SETH. In Section 5.4, we exploit the revealed connections between problems to show that it might also not be possible to replace the hardness assumption for Unbounded Knapsack with SETH. More precisely, we prove that there can be no deterministic reduction from SAT to Unbounded Knapsack that would rule out running time $\mathcal{O}(n^{1-\varepsilon}t)$ under the assumption of NSETH.

Then, in Section 5.7 we improve upon current state-of-the-art approximation for Min-Plus Convolution. As it turns out it gives an improved algorithm for Tree Sparsity, which we discuss in Section 5.8. Also, it turns out that our techniques automatically yield an improved approximation algorithm for 3SUM (see Section 5.9) which is optimal under some plausible assumption (see Section 5.10).

The preliminary version of the results in this Chapter was presented at *International Colloquium on Automata, Languages, and Programming* (ICALP 2019) [50]. The full version is published in *ACM Transactions on Algorithms* [51].

## 5.1 Basic reductions

**Theorem 5.1.1** (Unbounded Knapsack → Knapsack)**.** *A $T(n,t)$ algorithm for Knapsack implies an $\mathcal{O}(T(n\log t, t))$ algorithm for Unbounded Knapsack.*

*Proof.* Consider an instance of Unbounded Knapsack with capacity $t$ and the set of items given as weight-value pairs $((w_i, v_i))_{i\in\mathcal{I}}$. Construct an equivalent Knapsack instance with the same $t$ and the set of items $((2^j w_i, 2^j v_i))_{i\in\mathcal{I}, 0\le j\le \log t}$. Let $X = (x_i)_{i\in\mathcal{I}}$ be the list of multiplicities of items chosen in a solution to the Unbounded Knapsack problem. Of course, $x_i \le t$. Define $(x_i^j)_{0\le j\le \log t}$, $x_i^j \in \{0,1\}$ to be the binary representation of $x_i$. Then, the vector $(x_i^j)_{i\in\mathcal{I}, 0\le j\le \log t}$ induces a solution to Knapsack with the same total weight and value. The described mapping can be inverted. This implies the equivalence between the instances and proves the claim. □

We now consider the SuperAdditivity Testing problem. We start by showing that we can consider only the case of nonnegative monotonic sequences. This is a useful, technical assumption that simplifies the proofs.

**Lemma 5.1.2.** *Every sequence $(a[i])_{i=0,\dots,n-1}$ can be transformed in linear time to a nonnegative monotonic sequence $(a'[i])_{i=0,\dots,n-1}$ such that $a[i]$ is superadditive iff $a'[i]$ is superadditive.*

*Proof.* First, note that if $a[0] > 0$, then the sequence is not superadditive for $n > 0$ because $a[0] + a[i] > a[i]$. In the case where $a[0] \le 0$, the 0-th element does not influence the result of the algorithm. Thus, we can set $a'[0] = 0$ to ensure the nonnegativity of $a'$. Next, to guarantee monotonicity, we choose $C > 2\max_i\{|a[i]|\}$. Let

$$a'[i] = \begin{cases} 0, & \text{if } i = 0 \\ Ci + a[i], & \text{otherwise.} \end{cases}$$

Note that sequence $a'[i]$ is strictly increasing and nonnegative. Moreover, for $i, j > 0$,

$$\begin{aligned} a'[i] + a'[j] &\le a'[i+j] && \iff \\ C\cdot i + a[i] + C\cdot j + a[j] &\le C(i+j) + a[i+j] && \iff \\ a[i] + a[j] &\le a[i+j]. \end{aligned}$$

When $i$ or $j$ equals 0, then we have equality because $a'[0] = 0$. □

**Theorem 5.1.3** (SuperAdditivity Testing → Unbounded Knapsack)**.** *If Unbounded Knapsack can be solved in time $T(n,t)$, then SuperAdditivity Testing admits an algorithm with running time $\mathcal{O}(T(n,n)\log n)$.*

*Proof.* Let $(a[i])_{i=0}^{n-1}$ be a nonnegative monotonic sequence (see Lemma 5.1.2). Set $D = \sum_{i=0}^{n-1} a[i] + 1$, and construct an Unbounded Knapsack instance with the set of items $((i, a[i]))_{i=0}^{n-1}$ and $((2n-1-i, D-a[i]))_{i=0}^{n-1}$ with target $t = 2n-1$. It is always possible to obtain $D$ by taking two items $(i, a[i])$, $(2n-1-i, D-a[i])$ for any $i$. We claim that the answer to the constructed instance equals $D$ if and only if $a$ is superadditive.

If $a$ is not superadditive, then there are $i, j$ such that $a[i]+a[j] > a[i+j]$. Choosing $(i, a[i])$, $(j, a[j])$, $(2n-1-i-j, D-a[i+j])$ gives a solution with a value exceeding $D$.

Now, assume that $a$ is superadditive. Observe that any feasible knapsack solution may contain at most one item with a weight exceeding $n-1$. On the other hand, the optimal solution has to include one such item because the total value of the lighter ones is less than $D$. Therefore, the optimal solution contains an item $(2n-1-k, D-a[k])$ for some $k < n$. The total weight of the rest of the solution is at most $k$. As $a$ is superadditive, we can replace any pair $(i, a[i]), (j, a[j])$ with the item $(i+j, a[i+j])$ without decreasing the value of the solution. By repeating this argument, we end up with a single item lighter than $n$. The sequence $a$ is monotonic; thus, it is always profitable to replace these two items with the heavier one, as long as the load does not exceed $t$. We conclude that every optimal solution must be of the form $((k, a[k]), (2n-1-k, D-a[k]))$, which completes the proof. □

**Theorem 5.1.4** (Max-Plus Convolution UpperBound → SuperAdditivity Testing)**.** *If SuperAdditivity Testing can be solved in time $T(n)$, then* MaxConv UpperBound *admits an algorithm with running time $\mathcal{O}(T(n)\log n)$.*

*Proof.* We start by reducing the instance of MaxConv UpperBound to the case of nonnegative monotonic sequences (analogous to Lemma 5.1.2). Observe that condition $a[i] + b[j] \le c[i+j]$ can be rewritten as $(C + a[i] + Di) + (C + b[j] + Dj) \le 2C + c[i+j] + D(i+j)$ for any constants $C, D$. Hence, replacing sequences $(a[i])_{i=0}^{n-1}$, $(b[i])_{i=0}^{n-1}$, $(c[i])_{i=0}^{n-1}$ with $a'[i] = C + a[i] + Di$, $b'[i] = C + b[i] + Di$, $c'[i] = 2C + c[i] + Di$ leads to an equivalent instance. We can thus pick $C, D$ of magnitude $\mathcal{O}(W)$ to ensure that all elements are nonnegative and that the resulting sequences are monotonic. The values in the new sequences may increase to a maximum of $\mathcal{O}(nW)$.

Herein, we can assume the given sequences to be nonnegative and monotonic. Define $K$ to be the maximum value occurring in given sequences $a, b, c$. Construct a sequence $e$ of length $4n$ as follows. For $i \in [0, n-1]$, set

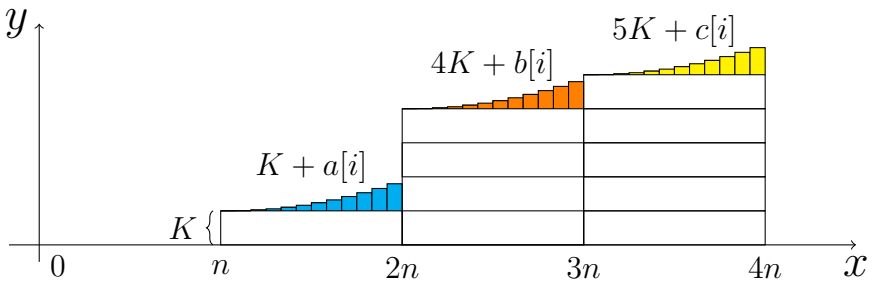



Figure 5.1: Graphical interpretation of the sequence $e$ in Theorem 5.1.4. The height of rectangles equals $K$.

$e[i] = 0$, $e[n+i] = K + a[i]$, $e[2n+i] = 4K + b[i]$, $e[3n+i] = 5K + c[i]$. If $a[i]+b[j] > c[i+j]$ exists for some $i, j$, then $e[n+i]+e[2n+j] > e[3n+i+j]$; therefore, $e$ is not superadditive. We now show that in any other case, $e$ must be superadditive.

Assume w.l.o.g. that there are $i$ and $j$ such that $i \leq j$. The case $i < n$ can be ruled out because it implies $e[i] = 0$ and $e[i] + e[j] \leq e[i+j]$ for any $j$, as $e$ is monotonic. If $i \geq 2n$, then $i + j \geq 4n$; thus, we can restrict to $i \in [n, 2n-1]$. For similar reasons, we can assume that $j < 3n$. Now, if $j \in [n, 2n-1]$, then $e[i] + e[j] \leq 4K \leq e[i+j]$. Finally, for $j \in [2n, 3n-1]$, superadditivity clearly corresponds to MaxConv UpperBound's defining condition. □

The proof of the reduction from Max-Plus Convolution to Max-Plus Convolution UpperBound was recently independently given in [18]. The technique was introduced by Vassilevska and Williams [154] to show a subcubic reduction from (min, +)-matrix multiplication for detecting a negative weight triangle in a graph.

**Theorem 5.1.5** (Max-Plus Convolution → Max-Plus Convolution UpperBound)**.** *A $T(n)$ algorithm for Max-Plus Convolution UpperBound implies an $\mathcal{O}\left(T(\sqrt{n})n \log n\right)$ algorithm for Max-Plus Convolution.*

*Proof.* Let us assume that we have access to an oracle solving the MaxConv UpperBound, i.e., checking whether $a \oplus^{\max} b \leq c$. First, we argue that by invoking this oracle $\log n$ times, we can find an index $k$ for which there exists a pair $i, j$ violating the superadditivity constraint, i.e., satisfying $a[i]+b[j] > c[k]$, where $k = i + j$ if such an index $k$ exists. Let $pre_k(s)$ be the $k$-element prefix of a sequence $s$. The inequality $pre_k(a) \oplus^{\max} pre_k(b) \leq pre_k(c)$ holds only for those $k$ that are less than the smallest value of $i + j$ with a broken constraint. We can use binary search to find the smallest $k$ for which the inequality does not hold. This introduces an overhead of factor $\log n$.

Next, we want to show that by using an oracle that finds one violated index, we can in fact find all violated indices. Let us divide $[0, n-1]$ into $m = \sqrt{n} + \mathcal{O}(1)$ intervals $I_0, I_2, \ldots I_m$ of equal length, except potentially for the last one. For each pair $I_x, I_y$, we can check whether $a[i] + b[j] \leq c[i+j]$ for all $i \in I_x$, $j \in I_y$ and find a violated constraint (if any exist) in time $T(\sqrt{n}) \log n$ by translating the indices to $[0, 2n/m] = [0, 2\sqrt{n} + \mathcal{O}(1)]$. After finding a pair $i, j$ that violates the superadditivity, we substitute $c[i+j] := K$, where $K$ is a constant exceeding all feasible sums, and continue analyzing the same pair. Once anomalies are no longer detected, we move on to the next pair. It is important to note that when an index $k$ violating superadditivity is set to $c[k] := K$, this value $K$ is also preserved for further calls to the oracle – in this way, we ensure that each violated index $k$ is reported only once.

For the sake of readability, we present a pseudocode (see Algorithm 2). The subroutine MaxConvDetectSingle returns the value of $i + j$ for a broken constraint or $-1$ if none exist. The notation $s^x$ stands for the subsequence of $s$ in the interval $I_x$. We assume that $c[i] = K$ for $i \geq n$.

**Algorithm 2** MaxConvDetectViolations$(a, b, c)$

```
1: for x = 0, ..., m − 1 do
2:     for y = 0, ..., m − 1 do
3:         k := 0
4:         while k ≥ 0 do
5:             k := MaxConvDetectSingle(a^x, b^y, c^{x+y} ∪ c^{x+y+1})
6:             if k ≥ 0 then
7:                 c[k] := K
8:                 violated[k] := true
9:             end if
10:        end while
11:    end for
12: end for
13: return violated[0, ..., n − 1]
```

The number of considered pairs of intervals equals $m^2 = \mathcal{O}(n)$. Furthermore, for each pair, every call to MaxConvDetectSingle except the last one is followed by setting a value of some element of $c$ to $K$. This can happen only once for each element; hence, the total number of repetitions is at most $n$. Therefore, the running time of the procedure MaxConvDetectViolations is $\mathcal{O}\left(T(\sqrt{n}) n \log n\right)$.

By running this algorithm, we learn for each $k \in [0, n-1]$ whether $c[k] > \max_{i \in [0,k]} a[i] + b[k-i]$. Then, we can again use binary search for each

coordinate simultaneously. After running the presented procedure $\log W$ times, the value of $c[k]$ will converge to $\max_{i\in[0,k]} a[i] + b[k-i]$ for every $k$. □

**Corollary 5.1.6.** *If there exists a truly subquadratic algorithm for Max-Plus Convolution, then it may be assumed to have $\widetilde{\mathcal{O}}(n)$ space dependency.*

*Proof.* Consider the Algorithm 2. It uses $\mathcal{O}(n)$ space to store the VIOLATED table containing the answer. The only other place where additional space might be required is the call to the MAXCONVDETECTSINGLE oracle. Note that each call runs in time $T(\sqrt{n})$, as the parameters are tables with $\mathcal{O}(\sqrt{n})$ elements. If Max-Plus Convolution has a truly subquadratic algorithm, then $T(\sqrt{n}) = \mathcal{O}(n^{1-\varepsilon/2})$, i.e., it is truly sublinear. Because the oracle cannot use polynomially more space than its running time, the calls to the oracle require at most linear space (up to polylogarithmic factors).

This means that the main space cost of Algorithm 2 is to store an answer in the table VIOLATED and yields $\widetilde{\mathcal{O}}(n)$ space dependency.

□

## 5.2 The reduction from Knapsack to Max-Plus Convolution

We start with a simple observation: for Unbounded Knapsack (a single item can be chosen multiple times), an $\widetilde{\mathcal{O}}(t^2+n)$ time algorithm can be obtained by using the standard dynamic programming $\mathcal{O}(nt)$ algorithm.

**Theorem 5.2.1.** *There exists an $\widetilde{\mathcal{O}}(t^2+n)$ time algorithm for the Unbounded Knapsack problem.*

*Proof.* Our algorithm starts by discarding all items with weight larger than $t$. Since we are considering the unbounded case, for a given weight, we can ignore all items except the one with the highest value, as we can always take more copies of the most valuable item among the ones of equal weight. We are left with at most $t$ items. Thus, using the standard $\mathcal{O}(nt)$ dynamic programming leads to a running time of $\widetilde{\mathcal{O}}(t^2+n)$. □

We show that from the perspective of the parameter $t$, this is the best we can hope for, unless $n$ appears in the complexity with an exponent higher than 2 or there is a breakthrough for the Max-Plus Convolution problem. In this section, we complement these results and show that a truly subquadratic algorithm for Max-Plus Convolution implies an $\widetilde{\mathcal{O}}(t^{2-\epsilon}+n)$ algorithm for

Knapsack. We follow Bringmann's [25] near-linear pseudo-polynomial time algorithm for Subset Sum and adapt it to the Knapsack problem. To do this, we need to introduce some concepts related to the Subset Sum problem from previous works. The key observation is that we can substitute the FFT in [25] with Max-Plus Convolution and consequently obtain an $\widetilde{\mathcal{O}}(T(t)+n)$ algorithm for Knapsack (where $T(n)$ is the time needed to solve Max-Plus Convolution).

### 5.2.1 Set of all subset sums

Let us recall that in the Subset Sum problem, we are given a set $S$ of $n$ integers together with a target integer $t$. The goal is to determine whether there exists a subset of $S$ that sums up to $t$.

Horowitz and Sahni [86] introduced the notion of the set of *all subset sums* that was later used by Eppstein [58] to solve the *Dynamic Subset Sum* problem. More recently, Koiliaris and Xu [102] used it to develop an $\widetilde{\mathcal{O}}(\sigma)$ algorithm for Subset Sum ($\sigma$ denotes the sum of all elements). Later, Bringmann [25] improved this algorithm to $\widetilde{\mathcal{O}}(n+t)$ ($t$ denotes the target number in the Subset Sum problem).

The set of *all subset sums* is defined as follows:

$$\Sigma(S) = \Big\{ \sum_{a \in A} a \mid A \subseteq S \Big\}.$$

For two sets $A, B \subseteq [0, u]$, the set $A \oplus B = \{a + b \mid a \in A, b \in B\}$ is their join, and $u$ is the upper bound of the elements $A$ and $B$. This join can be computed in time $\mathcal{O}(u \log u)$ by using the FFT. Namely, we write $A$ and $B$ as polynomials $f_A(x) = \sum_{i \in A} x^i$ and $f_B(x) = \sum_{i \in B} x^i$, respectively. Then, we can compute the polynomial $g = f_1 \cdot f_2$ in $\mathcal{O}(u \log u)$ time. Polynomial $g$ has a nonzero coefficient in front of the term $x^i$ iff $i \in A \oplus B$. We can also easily extract $A \oplus B$.

Koiliaris and Xu [102] noticed that if we want to compute $\Sigma(S)$ for a given $S$, we can partition $S$ into two sets: $S_1$ and $S_2$, recursively compute $\Sigma(S_1)$ and $\Sigma(S_2)$, and then join them using the FFT. Koiliaris and Xu [102] analyzed their algorithm using Lemma 5.2.2, which was later also used by Bringmann [25].

**Lemma 5.2.2** ([102], Observation 2.6)**.** *Let $g$ be a positive, superadditive (i.e., $\forall_{x,y} g(x+y) \geq g(x) + g(y)$) function. For a function $f(n, m)$ satisfying*

$$f(n, m) = \max_{m_1 + m_2 = m} \Big\{ f\Big(\frac{n}{2}, m_1\Big) + f\Big(\frac{n}{2}, m_2\Big) + g(m)) \Big\}$$

*we have that $f(n, m) = \mathcal{O}(g(m) \log n)$.*

### 5.2.2 Sum of all sets for Knapsack

We now adapt the notion of the sum of all sets to the Knapsack setting. Here, we use a data structure that, for a given capacity, stores the value of the best solution we can pack. This data structure can be implemented as an array of size $t$ that keeps the largest value in each cell (for comparison, $\Sigma(S)$ was implemented as a binary vector of size $t$). To emphasize that we are working with Knapsack, we use $\Pi(S)$ to denote the array of the values for the set of items $S$.

If we have two partial solutions $\Pi(A)$ and $\Pi(B)$, we can compute their join, denoted as $\Pi(A) \oplus^{\max} \Pi(B)$. A valid solution in $\Pi(A) \oplus^{\max} \Pi(B)$ of weight $t$ consists of a solution from $\Pi(A)$ and one from $\Pi(B)$ that sum up to $t$ (one of them can be 0). Hence, $\Pi(A) \oplus^{\max} \Pi(B)[k] = \max_{0 \leq i \leq k}\{\Pi(A)[k - i] + \Pi(B)[i]\}$. This product is the Max-Plus Convolution of array $\Pi(A)$ and $\Pi(B)$. We will use $\Pi(A) \oplus_t^{\max} \Pi(B)$ to denote the Max-Plus Convolution of $A$ and $B$ for domain $\{0, \ldots, t\}$.

To compute $\Pi(S)$, we can split $S$ into two equal-cardinality, disjoint subsets $S = S_1 \cup S_2$, recursively compute $\Pi(S_1)$ and $\Pi(S_2)$, and finally join them in $\mathcal{O}(T(\sigma))$ time ($\sigma$ is the sum of weights of all items). By Lemma 5.2.2, we obtain an $\mathcal{O}(T(\sigma) \log \sigma \log n)$ time algorithm (recall that the naive algorithm for Max-Plus Convolution works in $\mathcal{O}(n^2)$ time).

### 5.2.3 Retracing Bringmann's steps

In this section, we obtain an $\widetilde{\mathcal{O}}(T(t) + n)$ algorithm for Knapsack, which improves upon the $\widetilde{\mathcal{O}}(T(\sigma))$ algorithm from the previous section. In his algorithm [25] for Subset Sum, Bringmann uses two key techniques. First, *layer splitting* is based on a very useful observation that an instance $(Z, t)$ can be partitioned into $\mathcal{O}(\log n)$ layers $L_i \subseteq (t/2^i, t/2^{i-1}]$ (for $0 < i < \lceil \log n \rceil$) and $L_{\lceil \log n \rceil} \subseteq [0, t/2^{\lceil \log n \rceil - 1}]$. With this partition, we may infer that for $i > 0$, at most $2^i$ elements from the set $L_i$ can be used in any solution (otherwise, their cumulative sum would be larger than $t$). The second technique is an application of *color coding* [11] that results in a fast, randomized algorithm that can compute all solutions with a sum of at most $t$ using no more than $k$ elements. By combining those two techniques, Bringmann [25] developed an $\widetilde{\mathcal{O}}(t + n)$ time algorithm for Subset Sum. We now retrace both ideas and use them in the Knapsack context.

### Color Coding

We modify Bringmann's [25] color coding technique by using Max-Plus Convolution instead of FFT to obtain an algorithm for Knapsack. We first discuss the Algorithm 3, which can compute all solutions in $[0, t]$ that use at most $k$ elements with high probability. We start by randomly partitioning the set of items into $k^2$ disjoint sets $Z = Z_1 \cup \ldots \cup Z_{k^2}$. Algorithm 3 succeeds in finding a given solution if its elements are placed in different sets of the partition $Z$.

**Lemma 5.2.3.** *There exists an algorithm that computes an array $W$ in time $\mathcal{O}(T(t)k^2 \log(1/\delta))$ such that, for any $Y \subseteq Z$ with $|Y| \leq k$ and every weight $i \in [0, t]$, we have $\Pi(Y)[i] \leq W[i] \leq \Pi(Z)[i]$ with probability $\geq 1 - \delta$ for any constant $\delta \in (0, 1)$ (where $T(n)$ is the time needed to compute Max-Plus Convolution).*

**Algorithm 3** ColorCoding$(Z, t, k, \delta)$ (cf. [25, Algorithm 1]).

1: **for** $j = 1, \ldots, \left\lceil \log_{4/3}(1/\delta) \right\rceil$ **do**
2: randomly partition $Z = Z_1 \cup \ldots \cup Z_{k^2}$
3: $\mathbf{P}_j = Z_1 \oplus_t^{\max} \ldots \oplus_t^{\max} Z_{k^2}$
4: **end for**
5: **return** $W$, where $W[i] = \max_j \mathbf{P}_j[i]$

*Proof.* We show split $Z$ into $k^2$ parts: $Z_1 \cup \ldots \cup Z_{k^2}$. Here, $Z_i$ is an array of size $t$, and $Z_i[j]$ is the value of a single element (if one exists) with weight $j$ in $Z_i$ (in case of a conflict, we select a random one).

We claim that $Z_1 \oplus_t^{\max} \ldots \oplus_t^{\max} Z_{k^2}$ contains solutions at least as good as those that use $k$ items (with high probability). We use the same argument as in [25]. Assume that the best solution uses the set $Y \subseteq Z$ of items and $|Y| \leq k$. The probability that all items of $Y$ are in different sets of the partition is the same as the probability that the second element of $Y$ is in a different set than the first one, the third element is in a different set than the first and second item, etc. That is:

$$\frac{k^2-1}{k^2} \cdot \frac{k^2-2}{k^2} \cdots \frac{k^2-(|Y|-1)}{k^2} \geq \left(1 - \frac{(|Y|-1)}{k^2}\right)^{|Y|} \geq \left(1 - \frac{1}{k}\right)^{k} \geq \left(\frac{1}{2}\right)^{2} = \frac{1}{4}.$$

By repeating this process $\mathcal{O}(\log(\frac{1}{\delta}))$ times, we obtain the correct solution with a probability of at least $1 - \delta$. Also, to compute Max-Plus Convolution, we need $k^2$ repetitions. Hence, we obtain an $\mathcal{O}(T(t)k^2 \log(1/\delta))$ time algorithm. □

### Layer Splitting

We can split our items into $\log n$ layers. Layer $L_i$ is the set of items with weights in $(t/2^i, t/2^{i-1}]$ for $0 < i < \lceil \log n \rceil$; the last layer $L_{\lceil \log n \rceil}$ has items with weights in $[0, t/2^{\lceil \log n \rceil - 1}]$. With this, we can be sure that only $2^i$ items from the layer $i$ can be chosen for a solution. If we can quickly compute $\Pi(L_i)$ for all $i$, then it suffices to compute their Max-Plus Convolution $\mathcal{O}(\log n)$ times. We now show how to compute $\Pi(L_i)$ in $\widetilde{\mathcal{O}}(T(t) + n)$ time using color coding.

**Lemma 5.2.4.** *For all $i$, there exists an algorithm that, for $L_i \subseteq (\frac{t}{2^i}, \frac{t}{2^{i-1}}]$ and for all $\delta \in (0, 1/4]$, computes $\Pi(L_i)$ in $\mathcal{O}(T(t \log t \log^3(2^{i-1}/\delta)))$ time, where each entry of $\Pi(L_i)$ is correct with a probability of at least $1 - \delta$.*

**Algorithm 4** ColorCodingLayer$(L, t, i, \delta)$ (cf. [25, Algorithm 3]).

1: $l = 2^i$
2: **if** $l < \log(l/\delta)$ **then return** ColorCoding$(L, t, l, \delta)$
3: $m = l/\log(l/\delta)$ rounded up to the next power of 2
4: randomly partition $L = A_1 \cup \ldots \cup A_m$
5: $\gamma = 6 \log(l/\delta)$
6: **for** $j = 1, \ldots, m$ **do**
7: $\quad \mathbf{P}_j =$ ColorCoding$(A_j, 2\gamma t/l, \gamma, \delta/l)$
8: **end for**
9: **for** $h = 1, \ldots, \log m$ **do**
10: $\quad$ **for** $j = 1, \ldots, m/2^h$ **do**
11: $\quad\quad \mathbf{P}_j = \mathbf{P}_{2j-1} \oplus^{\max}_{2^h \cdot 2\gamma t/l} \mathbf{P}_{2j}$
12: $\quad$ **end for**
13: **end for**
14: **return** $\mathbf{P}_1$

*Proof.* We use the same arguments as in [25, Lemma 3.2]. First, we split the set $L$ into $m$ disjoint subsets $L = A_1 \cup \ldots \cup A_m$ (where $m = l/\log(l/\delta)$). Then, for every partition, we compute $\Pi(A_i)$ using $\mathcal{O}(\log(l/\delta))$ items and probability $\delta/l$ using Lemma 5.2.3. For every $A_i$, $\mathcal{O}(T(\log(l)t/l) \log^3(l/\delta))$ time is required. Hence, for all $A_i$, we need $\mathcal{O}(T(t) \log^3(l/\delta))$ time, as Min-Plus Convolution needs at least linear time $T(n) = \Omega(n)$.

Ultimately, we need to combine arrays $\Pi(A_i)$ in a "binary tree way". In the first round, we compute $\Pi(A_1) \oplus^{\max} \Pi(A_2), \Pi(A_3) \oplus^{\max} \Pi(A_4), \ldots, \Pi(A_{m-1}) \oplus^{\max} \Pi(A_m)$. Then, in the second round, we join the products of the first round in a similar way. We continue until we have joined all subsets. This process

yields us significant savings over just computing $\Pi(A_1) \oplus^{\max} \ldots \oplus^{\max} \Pi(A_m)$ because in round $h$, we need to compute Max-Plus Convolution with numbers of order $\mathcal{O}(2^h t \log(l/\delta)/l)$, and there are at most $\log m$ rounds. The complexity of joining them is as follows:

$$\sum_{h=1}^{\log m} \frac{m}{2^h} T(2^h \log(l/\delta) t/l) \log t) = \mathcal{O}(T(t \log t) \log m).$$

Overall, we determine that the time complexity of the algorithm is $\mathcal{O}(T(t \log t) \log^3(l/\delta))$ (some logarithmic factors could be omitted if we assume that there exists $\epsilon > 0$ such that $T(n) = \Omega(n^{1+\epsilon})$).

The correctness of the algorithm is based on [25, Claim 3.3]. We take a subset of items $Y \subseteq L$ and let $Y_j = Y \cap A_j$. Claim 3.3 in [25] says that $\mathbb{P}\left[[\,] |Y_j| \geq 6 \log(l/\delta)\right] \leq \delta/l$. Thus, we can run ColorCoding procedure for $k = 6 \log(l/\delta)$ and still guarantee a sufficiently high probability of success. □

**Theorem 5.2.5** (Knapsack → Max-Plus Convolution)**.** *If Max-Plus Convolution can be solved in $T(n)$ time, then Knapsack can be solved in time $\mathcal{O}(T(t \log t) \log^3(n/\delta) \log n)$ with a probability of at least $1 - \delta$.*

**Algorithm 5** Knapsack$(Z, t, \delta)$ (cf. [25, Algorithm 2]).

1: split $Z$ into $L_i = Z \cap (t/2^i, t/2^{i-1}]$ for $i = 1, \ldots, \lceil \log n \rceil - 1$, and $L_{\lceil \log n \rceil} = Z \cap [0, t/2^{\lceil \log n \rceil - 1}]$
2: $\mathbf{W} = \emptyset$
3: **for** $i = 1, \ldots, \lceil \log n \rceil$ **do**
4: $\quad \mathbf{P}_i = \textsc{ColorCodingLayer}(L_i, t, i, \delta/\lceil \log n \rceil)$
5: $\quad \mathbf{W} = \mathbf{W} \oplus^{\max} \mathbf{P}_i$
6: **end for**
7: **return W**

*Proof.* To obtain an algorithm for Knapsack, as mentioned before, we need to split $Z$ into disjoint layers $L_i = Z \cap (t/2^i, t/2^{i-1}]$ and $L_{\lceil \log n \rceil} = Z \cap [0, t/2^{\lceil \log n \rceil - 1}]$. Then, we compute $\Pi(L_i)$ for all $i$ and join them using Max-Plus Convolution. We present the pseudocode in Algorithm 5. It is based on [25, Algorithm 2]. Overall, $\mathcal{O}(T(t \log t) \log^3(n/\delta) \log n + T(t) \log n) = \mathcal{O}(T(t \log t) \log^3(n/\delta) \log n)$ time is required. □

Koiliaris and Xu [102] considered a variant of Subset Sum where one needs to check if there exists a subset that sums up to $k$ for all $k \in [0, t]$. Here,

we note that a similar extension for Knapsack is also equivalent to Max-Plus Convolution.

**Corollary 5.2.6** ($0/1Knapsack^+ \rightarrow$ Max-Plus Convolution)**.** *If Max-Plus Convolution can be solved in $T(n)$ time, then $0/1Knapsack^+$ can be solved in $\mathcal{O}(T(t\log t)\log^3(tn/\delta)\log n)$ time with a probability of at least $1-\delta$.*

Algorithm 5 returns an array $\Pi(Z)$, where each entry $z \in \Pi(Z)$ is optimal with probability $1-\delta$. Now, if we want to obtain the optimal solution for all knapsack capacities in $[1,t]$, we need to increase the success probability to $1-\frac{\delta}{t}$ so that we can use the union bound. Consequently, in this case, a single entry is faulty with a probability of at most $\delta/t$, and we can upper bound the event, where at least one entry is incorrect by $\frac{\delta}{t}t = \delta$. This introduces an additional polylog$(t)$ factor in the running time.

Finally, for completeness, we note that $0/1Knapsack^+$ is more general than Knapsack. $0/1Knapsack^+$ returns a solution for all capacities $\leq t$. However, in the Knapsack problem, we are interested only in a capacity equal to exactly $t$.

**Corollary 5.2.7** (Knapsack $\rightarrow 0/1Knapsack^+$)**.** *If $0/1Knapsack^+$ can be solved in $T(t,n)$ time, then Knapsack can be solved in $\widetilde{\mathcal{O}}(T(t,n))$ time.*

The next corollary follows from the ring of reductions.

**Corollary 5.2.8.** *An $\widetilde{\mathcal{O}}((n+t)^{2-\varepsilon})$ time algorithm for Knapsack implies an $\widetilde{\mathcal{O}}(t^{2-\varepsilon'}+n)$ time algorithm for $0/1Knapsack^+$.*

## 5.3 Other problems related to Min-Plus Convolution

### 5.3.1 Maximum Consecutive Subsums Problem

The Maximum Consecutive Subsums Problem (MCSP) is, to the best of our knowledge, the first problem explicitly proven to be nontrivially subquadratically equivalent to MinConv [104]. In this section, we show the reduction from MCSP to Max-Plus Convolution for completeness. Moreover, we present the reduction in the opposite direction, which, in our opinion, is simpler than the original one.

**Theorem 5.3.1** (MCSP $\rightarrow$ Max-Plus Convolution)**.** *If Max-Plus Convolution can be solved in time $T(n)$, then MCSP admits an algorithm with running time $\mathcal{O}(T(n))$.*

*Proof.* Let $(a[i])_{i=0}^{n-1}$ be the input sequence. Construct sequences of length $2n$ as follows: $b[k] = \sum_{i=0}^{k} a[i]$ for $k < n$ and $c[k] = -\sum_{i=0}^{n-k-1} a[i]$ for $k \leq n$ (empty sum equals 0); otherwise, $b[k] = c[k] = -D$, where $D$ is two times larger than any partial sum. Observe that

$$(b \oplus^{\max} c)[n+k-1] = \max_{\substack{0 \leq j < n \\ 0 \leq n+k-j-1 \leq n}} \sum_{i=0}^{j} a[i] - \sum_{i=0}^{j-k} a[i] = \max_{k-1 \leq j < n} \sum_{i=j-k+1}^{j} a[i]. \tag{5.1}$$

Thus, we can determine the maximum consecutive sum for each length $k$ after performing Max-Plus Convolution. □

**Theorem 5.3.2** (SuperAdditivity Testing → MCSP)**.** *If MCSP can be solved in time $T(n)$, then SuperAdditivity Testing admits an algorithm with running time $\mathcal{O}(T(n))$.*

*Proof.* Let $(a[i])_{i=0}^{n-1}$ be the input sequence and $b[i] = a[i+1] - a[i]$. The superadditivity condition $a[k] \leq a[k+j] - a[j]$ (for all possible $k, j$) can be translated into $a[k] \leq \min_{0 \leq j < n-k} \sum_{i=j}^{k+j-1} b[i]$ (for all $k$). Thus, computing the MCSP vector on $(-b[i])_{i=0}^{n-2}$ is sufficient to verify whether the above condition holds. □

### 5.3.2 Tree Sparsity

**Theorem 5.3.3** (Tree Sparsity → Max-Plus Convolution)**.** *If Max-Plus Convolution can be solved in time $T(n)$ and the function $T$ is superadditive, then Tree Sparsity admits an algorithm with running time $\mathcal{O}\left(T(n) \log^2 n\right)$.*

*Proof.* We take advantage of the heavy-light decomposition introduced by Sleator and Tarjan [140]. This technique has been utilized by Backurs et al. [18] to transform a nearly linear PTAS for Max-Plus Convolution to a nearly linear PTAS for Tree Sparsity.

We decompose a tree into a set of paths (which we call *spines*) that will start from a *head*. First, we construct a *spine* with a *head* $s_1$ at the root of the tree. We define $s_{i+1}$ as the child of $s_i$ for a larger subtree (in case of a draw, we choose any child) and the last node in the spine as a leaf. The remaining children of node $s_i$ become heads for analogous spines such that the whole tree is covered. Note that every path from a leaf to the root intersects at most $\log n$ spines because each spine transition doubles the subtree size.

Similar to [18] for a node $v$ with a subtree of size $m$, we want to compute the *sparsity vector* $U = (U[0], U[1], \ldots, U[m])$, where the index $U[i]$ represents the weight of the heaviest subtree rooted at $v$ with size $i$. We compute sparsity vectors for all heads of spines in the tree recursively. Let $(s_i)_{i=1}^{\ell}$ be

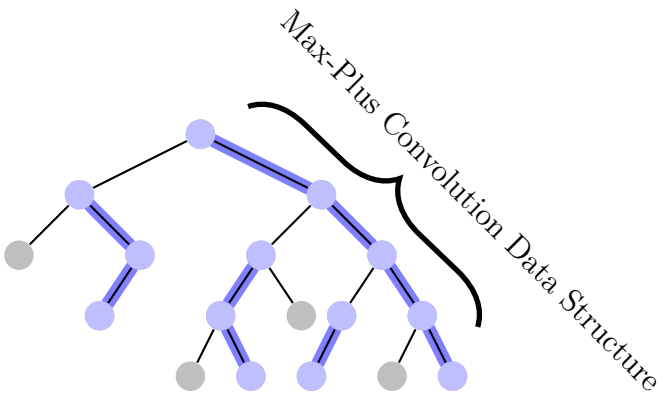



Figure 5.2: Schema of spine decomposition [18]. Blue edges represent edges on the spine. For each spine, we build an efficient data structure that uses Max-Plus Convolution (curly brackets). There are at most $\mathcal{O}(\log n)$ different spines on any path from a leaf to the root.

a spine with a head $v$, and for all $i$, let $U^i$ indicate the sparsity vector for the child of $s_i$ that is a head (i.e., the child with the smaller subtree). If $s_i$ has less than two children, then $U^i$ is a zero vector.

For an interval $[a, b] \subseteq [1, \ell]$, let $U^{a,b} = U^a \oplus^{\max} U^{a+1} \oplus^{\max} \cdots \oplus^{\max} U^b$, and let $Y^{a,b}[k]$ be a vector such that for all $k$, $Y^{a,b}[k]$ is the weight of a subtree of size $k$ rooted at $s_a$ and not containing $s_{b+1}$ (if it exists). Let $c = \left\lfloor \frac{a+b}{2} \right\rfloor$. The $\oplus^{\max}$ operator is associative; hence, $U^{a,b} = U^{a,c} \oplus^{\max} U^{c+1,b}$. To compute the vector $Y^{a,b}$, we consider two cases, depending on whether the optimal subtree contains $s_{c+1}$.

$$\begin{aligned} Y^{a,b}[k] &= \max\left[Y^{a,c}[k], \quad \sum_{i=a}^{c} w(s_i) + \max_{k_1+k_2=k-(c-a+1)} \Big(U^{a,c}[k_1] + Y^{c+1,b}[k_2]\Big)\right] \\ &= \max\left[Y^{a,c}[k], \quad \sum_{i=a}^{c} w(s_i) + \Big(U^{a,c} \oplus^{\max} Y^{c+1,b}\Big)\Big[k-(c-a+1)\Big]\right] \end{aligned}$$

Recall, that $w : V(T) \to \mathbb{N}_{\geq 0}$ is the weight function from the definition of the problem (see Section 1.3.1). Using the presented formulas, we reduce the problem of computing $X^v = Y^{1,\ell}$ to subproblems for intervals $[1, \frac{\ell}{2}]$ and $[\frac{\ell}{2} + 1, \ell]$, and we merge the results with two $(\max, +)$-convolutions. Proceeding further, we obtain $\log \ell$ levels of recursion, where the sum of convolution sizes on each level is $\mathcal{O}(m)$, which results in a total running time of $\mathcal{O}\left(T(m) \log m\right)$ (recall that $T$ is superadditive).

The heavy-light decomposition guarantees that there are at most $\mathcal{O}(\log n)$

different spines on a path from a leaf to the root. Moreover, we compute sparsity vectors for all heads of the spine, with at most $\log n$ levels of recursion. In each recursion, we execute the Max-Plus Convolution procedure. Hence, we obtain a running time of $\mathcal{O}(T(n)\log^2 n)$. □

### 5.3.3 $l_\infty$-Necklace Alignment

In this section, we study the $l_\infty$-Necklace Alignment alignment problem, which has been shown to be reducible to Min-Plus Convolution [24]. Even though we were not able to prove it as equivalent to Min-Plus Convolution, we have observed that $l_\infty$-Necklace Alignment is tightly connected to the $(\min,+)$-convolution, which leads to a reduction from a related problem – Max-Plus Convolution LowerBound. This opens an avenue for expanding the class of problems equivalent to Min-Plus Convolution; however, it turns out that we first need to better understand the nondeterministic complexity of Min-Plus Convolution. We elaborate on these issues in this and the following section.

**Theorem 5.3.4** (Max-Plus Convolution LowerBound $\rightarrow$ $l_\infty$-Necklace Alignment)**.** *If $l_\infty$-Necklace Alignment can be solved in time $T(n)$, then Max-Plus Convolution LowerBound admits an algorithm with running time $\mathcal{O}(T(n)\log n)$.*

*Proof.* Let $a, b, c$ be the input sequences for Max-Plus Convolution LowerBound. A *combination* is the sum of any choice of $m$ elements from these sequences. More formally:

**Definition 5.3.5** (combination)**.** *A* combination *of length $m$ is a sum:*

$$f_1 \cdot e_1[k_1] + f_2 \cdot e_2[k_2] + \ldots f_m \cdot e_m[k_m],$$

*where $e_i \in \{a, b, c\}$, $f_i \in \{-1, 1\}$ and $k_i \in \{0, \ldots, n-1\}$.*
*The* order *of this combination is as follows:*

$$\sum_{i=1}^{m} f_i \cdot k_i.$$

We can assume the following properties of the input sequences w.l.o.g.

1. *We may assume that the sequences are nonnegative and that $a[i] \leq c[i]$ for all $i$.* To guarantee this, we add $C_1$ to $a$, $C_1 + C_2$ to $b$, and $2C_1 + C_2$ to $c$ for appropriate positive constants $C_1, C_2$.

2. *We can assume that the combinations of order $\leq n$ that contain the last element of sequence $b$ with a positive coefficient are positive.* We can

achieve this property by artificially appending any $b[n]$ that is larger than the sum of all elements. Note that since it is the last element, it does not influence the result of the Max-Plus Convolution LowerBound instance.

3. *Any combination of positive order and length bounded by $L$ has a non-negative value.* One can guarantee this by adding a linear function $Di$ to all sequences. As the order of the combination is positive, the factors at $D$ sum up to a positive value. It suffices to choose $D$ equal to the maximum absolute value of an element times a parameter $L$ that will be set to 10. Note that previous inequalities compare combinations of the same order, and so they remain unaffected.

These transformations might increase the values of the elements to $\mathcal{O}(nWL^2)$. Let $B = b[n]$, $B_1 = b[n-1]$, $B_2 = b[n] - b[1]$. We define necklaces $x, y$ of length $2B$ with $N = 2n$ beads each.

$$
\begin{array}{lccccccccc}
x = \Big( & a[0], & a[1], & \ldots, & a[n-1], & B + c[0], & B + c[1], & \ldots, & B + c[n-2], & B + c[n-1] \Big), \\
y = \Big( & B_1 - b[n-1], & B_1 - b[n-2], & \ldots, & B_1 - b[0], & B + B_2 - b[n-1], & B + B_2 - b[n-2], & \ldots, & B + B_2 - b[1], & 2B \Big).
\end{array}
$$

Property (3) implies monotonicity of the sequences because for any $0 \leq i < j \leq n$, the combination $a[j] - a[i]$ is greater than zero.

Let $d(x[i], y[j])$ be the forward distance between $x[i]$ and $y[j]$, i.e., $y[j] - x[i]$ plus the length of the necklaces if $j < i$. For all $k$, define $M_k$ to be $\max_{i \in [0,N)} d\Big(x[i], y[(k+i) \pmod{2n}]\Big) - \min_{i \in [0,N)} d\Big(x[i], y[(k+i) \pmod{2n}]\Big)$. In this setting, [24, Fact 5] says that for a fixed $k$, the optimal solution has a value of $\frac{M_k}{2}$.

We want to show that for $k \in [0, n)$, the following holds:

$$
\begin{aligned}
\min_{i \in [0,2n)} d\Big(x[i], y[(k+i) \pmod{2n}]\Big) &= B_1 - \max_{i+j=n-k-1} (a[i] + b[j]), \\
\max_{i \in [0,2n)} d\Big(x[i], y[(k+i) \pmod{2n}]\Big) &= B - c[n-k-1].
\end{aligned}
$$

There are five types of connections between beads (see Figure 5.3).

$$
d\Big(x[i], y[(k+i) \pmod{2n}]\Big) =
\begin{cases}
B_1 - a[i] - b[n-k-1-i] & i \in [0, n-k-1], \quad \text{(I)} \\
B + B_2 - a[i] - b[2n-k-1-i] & i \in [n-k, n-1], \quad \text{(II)} \\
B_2 - b[2n-k-1-i] - c[i-n] & i \in [n, 2n-k-2], \quad \text{(III)} \\
B - c[n-k-1] & i = 2n-k-1, \quad \text{(IV)} \\
B + B_1 - b[3n-k-1-i] - c[i-n] & i \in [2n-k, 2n-1]. \quad \text{(V)}
\end{cases}
$$

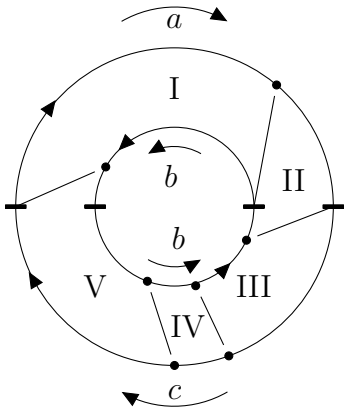



Figure 5.3: Five areas that correspond to the five types of connections between beads. The inner circle represents two repetitions of the sequence $b$. The outer circle consists of the sequence $a$ and then the sequence $c$.

All formulas form combinations of length bounded by 5; thus, we can apply properties (2) and (3). Observe that the order of each combination equals $k$, except for $i = 2n - k - 1$, where the order is $k + 1$. Using property (3), we reason that $B - c[n - k - 1]$ is indeed the maximal forward distance. We now show that the minimum lies within the group (I). First, note that these are the only combinations with no occurrences of $b[n]$. We claim that every distance in group (I) is upperbounded by all distances in other groups. This is clear for group (IV) because the orders differ. For other groups, we can use property (2), as the combinations in question have the same order and only the one not in group (I) contains $b[n]$.

For $k < n$, the condition $M_k < B - B_1$ is equivalent to $c[n - k - 1] > \max_{i+j=n-k-1}(a[i] + b[j])$. If such a $k$ exists, i.e., the answer to Max-Plus Convolution LowerBound for sequences $a, b, c$ is NO, then $\min_k M_k < B - B_1$ and the return value is less than $\frac{1}{2}(B - B_1)$.

Finally, we need to prove that $M_k \geq B - B_1$ for all $k$ if such a $k$ does not exist. We have already verified this to be true for $k < n$. Each matching for $k \geq n$ can be represented as swapping sequences $a$ and $c$ inside the necklace $x$, developed via an index shift of $k - n$. The two halves of the necklace $x$ are analogous; thus, all prior observations of the matching structure remain valid.

If the answer to Max-Plus Convolution LowerBound for sequences $a, b, c$ is YES, then $\forall_{k\in[0,n)}\exists_{i+j=k} a[i]+b[j] \geq c[k]$. Property (1) guarantees that $a \leq c$; thus, we conclude that $\forall_{k\in[0,n)}\exists_{i+j=k} c[i] + b[j] \geq a[i] + b[j] \geq c[k] \geq a[k]$, and by the same argument as before, the cost of the solution is at least $\frac{1}{2}(B - B_1)$. □

Observe that both $l_\infty$-Necklace Alignment and Max-Plus Convolution LowerBound admit simple linear nondeterministic algorithms. For Max-Plus Convolution LowerBound, it is sufficient to either assign each $k$ a single condition $a[i] + b[k-i] \geq c[k]$ that is satisfied or to nondeterministically guess a value of $k$ for which no inequality holds. For $l_\infty$-Necklace Alignment, we define a decision version of the problem by asking if there is an alignment of the value bounded by $K$ (the problem is self-reducible via binary search). For positive instances, the algorithm simply nondeterministically guesses $k$, inducing an optimal solution. For negative instances, $M_k > 2K$ must hold for all $k$. Therefore, it suffices to nondeterministically guess for each $k$ a pair $i, j$ such that $d\Big(x[i], y[(k+i) \pmod n]\Big) - d\Big(x[j], y[(k+j) \pmod n]\Big) > 2K$.

In Section 5.4, we will show that Max-Plus Convolution UpperBound admits an $\mathcal{O}\left(n^{1.5}\text{polylog}(n)\right)$ nondeterministic algorithm (see Lemma 5.4.1) so, in fact, there is no obstacle to the existence of a subquadratic reduction from Max-Plus Convolution LowerBound to Max-Plus Convolution UpperBound. However, the nondeterministic algorithm for 3SUM exploits techniques significantly different from ours, including modular arithmetic. A potential reduction would probably need to rely on some different structural properties of Max-Plus Convolution.

## 5.4 Nondeterministic algorithms

Recently, Abboud et al. [3] proved that the running time for the SUBSET SUM problem cannot be improved to $\mathcal{O}(t^{1-\varepsilon}2^{o(n)})$, assuming the SETH. It is tempting to look for an analogous lower bound for KNAPSACK that would make the $\mathcal{O}(nt)$-time algorithm tight. In this section, we take advantage of the nondeterministic lens introduced by Carmosino et al. [34] to argue that the existence of this lower bound for Unbounded Knapsack is unlikely.

We recall that by a time complexity of a nondeterministic algorithm, we refer to a bound on running times for both nondeterministic and co-nondeterministic routines determining whether an instance belongs to the language. Assuming the Nondeterministic Strong Exponential Time Hypothesis (NSETH), we cannot break the $\mathcal{O}(2^{(1-\varepsilon)n})$ barrier for SAT even with nondeterministic algorithms.

The informal reason to rely on the NSETH is that if we decide to base lower bounds on the SETH, then we should believe that SAT is indeed a very hard problem that does not admit any hidden structure that has eluded researchers so far. On the other hand, the NSETH can be used to rule out deterministic reductions from SAT to problems with nontrivial nondeterministic algorithms. This allows us to argue that in some situations basing

a hardness theory on the SETH can be a bad idea. Moreover, disproving the NSETH would imply nontrivial lower bounds on circuit sizes for $\mathbf{E}^{\mathbf{NP}}$ [34].

We present a nondeterministic algorithm for the decision version of Unbounded Knapsack with running time $\mathcal{O}(t\sqrt{n}\log^3(W))$, where $W$ is the target value. This means that a running time $\mathcal{O}(n^{1-\varepsilon}t)$ for Unbounded Knapsack cannot be ruled out with a deterministic reduction from SAT, under the assumption of the NSETH (for small $\varepsilon < \frac{1}{2}$).

We begin with an observation that a nontrivial nondeterministic algorithm for 3SUM entails a similar result for Max-Plus Convolution UpperBound.

**Lemma 5.4.1.** *Max-Plus Convolution UpperBound admits a nondeterministic* $\mathcal{O}\left(n^{1.5}\text{polylog}(n)\right)$*-time algorithm.*

*Proof.* By combining Theorem 5.5.1 (which involves a reduction from Max-Plus Convolution UpperBound to 3sumConv), the deterministic (i.e., nonrandomized) reduction from 3sumConv to 3SUM [124], and the nondeterministic $\mathcal{O}\left(n^{1.5}\text{polylog}(n)\right)$-time algorithm for 3SUM from [34, Lemma 5.8], we obtain an analogous algorithm for Max-Plus Convolution UpperBound. □

In the next step, we require a more careful complexity analysis of the nondeterministic algorithm for 3UM developed by Carmosino et al. [34, Lemma 5.8]. Essentially, we claim that the running time can be bounded by $\mathcal{O}(\sqrt{n_1 n_2 n_3}\log^2(W))$, where $n_1, n_2, n_3$ are sizes of the input sets. This is just a reformulation of the original proof, where an $\mathcal{O}(n^{1.5})$ nondeterministic time algorithm is given, which we have presented in the Section 5.6 for completeness.

In the decision version of Unbounded Knapsack, we are additionally given a threshold $W$, and we need to determine whether there is a multiset of items with a total weight of at most $t$ and a total value of at least $W$.

**Theorem 5.4.2.** *The decision version of Unbounded Knapsack admits an* $\mathcal{O}(t\sqrt{n}\log^3(W))$ *nondeterministic algorithm.*

*Proof.* We can assume that $n \leq t$. If we are given a YES-instance, then we can just nondeterministically guess the solution and verify it in $\mathcal{O}(t)$ time.

To show that an instance admits no solution, we nondeterministically guess a proof involving an array $(a[k])_{k=0}^{t}$ such that $a[k]$ is an upper bound for the total value of items with weights summing to at most $k$. To verify the proof, we need to check that $a[0] = 0$, $a[t] < W$, $a$ is nondecreasing, and, for each $k$ and each item $(w_i, v_i)$, $a[k] + v_i \leq a[k + w_i]$ holds. Let $(b[k])_{k=0}^{t}$ be a sequence defined as follows: if there is an item with $w_i = k$, then we set $b[k] = v_i$ (if there are multiple items with the same weight, we choose the most valuable one) and 0 otherwise. The latter condition is

equivalent to determining if $a \oplus^{\max} b \leq a$, which is an instance of MaxConv UpperBound with elements bounded by $W$.

Note that the sequence $b$ contains only $n$ nonzero elements. After we have verified (in $\mathcal{O}(t)$ time) that $a$ is nondecreasing, we know that $b[j] = 0$ implies $a[i] + b[j] \leq a[i + j]$. This means that we can neglect the zero values in sequence $b$ when applying the reduction in Theorem 5.5.1. After performing the reduction, we obtain $\mathcal{O}(\log W)$ instances of 3sumConv with sequences $x, y, z$ of length $t$ but with the additional knowledge that there are only $n$ indices $j$ such that $x[i]+y[j] > z[i+j]$ might hold. In the end, we perform a deterministic reduction from 3sumConv to 3sum in time $\mathcal{O}(t)$ [124]. Since we can omit all but $n$ indices in sequence $y$, we obtain $\mathcal{O}(\log W)$ instances of 3sum with set sizes of $t, n$, and $t$. The claim follows by applying Lemma 5.6.1 and the fact that nonrandomized reductions preserve the nondeterministic running time. □

From [34, Corollary 5.2] and the nondeterministic algorithm from Theorem 5.4.2, it follows that the reduction from any SETH-hard problem to Unbounded Knapsack is unlikely:

**Corollary 5.4.3.** *Under the NSETH, there is no deterministic (fine-grained) reduction from the SETH to solving Unbounded Knapsack in time $\mathcal{O}(n^{0.5+\gamma} \cdot t)$ for any $\gamma > 0$.*

For a natural (but rather technical) definition of *fine-grained reduction*, see [34, Definition 3.1].

## 5.5 Reduction to 3SUM

In this section, we show a connection between Max-Plus Convolution and the 3sum conjecture.

So far, we showed an equivalence between Max-Plus Convolution and Max-Plus Convolution UpperBound (see Theorem 1.3.11). Also, it is known that the 3sumConv problem is subquadratically equivalent to 3sum [124]. Hence, the following theorem suffices.

**Theorem 5.5.1** (Max-Plus Convolution UpperBound → 3sumConv)**.** *If 3sumConv can be solved in time $T(n)$, then Max-Plus Convolution UpperBound admits an algorithm with running time $\mathcal{O}(T(n))$.*

The proof heavily utilizes [161, Proposition 3.4, Theorem 3.3], which we present here for completeness. $\text{pre}_i(x)$ denotes the binary prefix of $x$ of length

$i$, where the most significant bit is considered the first. In the original statement (Proposition 3.4 [161]), the prefixes are alternately treated as integers or strings. We modify the notation slightly to work only with integers.

**Lemma 5.5.2** (Proposition 3.4 [161])**.** *For three integers $x, y, z$, we have that $x + y > z$ iff one of the following holds:*

1. *there exists a $k$ such that $\mathrm{pre}_k(x) + \mathrm{pre}_k(y) = \mathrm{pre}_k(z) + 1$,*
2. *there exists a $k$ such that*

$$\begin{aligned} \mathrm{pre}_{k+1}(x) &= 2 \cdot \mathrm{pre}_k(x) + 1, && (5.2)\\ \mathrm{pre}_{k+1}(y) &= 2 \cdot \mathrm{pre}_k(y) + 1, && (5.3)\\ \mathrm{pre}_{k+1}(z) &= 2 \cdot \mathrm{pre}_k(z), && (5.4)\\ \mathrm{pre}_k(z) &= \mathrm{pre}_k(x) + \mathrm{pre}_k(y). && (5.5) \end{aligned}$$

*Proof of Theorem 5.5.1.* We translate the inequality $a[i]+b[j] > c[i+j]$ from Max-Plus Convolution UpperBound to an alternative of $2 \log W$ equations. For each $0 \leq k \leq \log W$, we construct two instances of 3sumConv related to the conditions in Lemma 5.5.2. For the first condition, we create sequences $a^k[j] = \mathrm{pre}_k(a[j])$, $b^k[j] = \mathrm{pre}_k(b[j])$, $c^k[j] = \mathrm{pre}_k(c[j]) + 1$. For the second one, we choose a value of $D$ that is two times larger than the absolute value of any element and set

$$\begin{aligned} \tilde{a}^k[j] &= \begin{cases} \mathrm{pre}_k(a[j]) & \text{if } \mathrm{pre}_{k+1}(a[j]) = 2 \cdot \mathrm{pre}_k(a[j]) + 1,\\ -D & \text{otherwise,} \end{cases}\\ \tilde{b}^k[j] &= \begin{cases} \mathrm{pre}_k(b[j]) & \text{if } \mathrm{pre}_{k+1}(b[j]) = 2 \cdot \mathrm{pre}_k(b[j]) + 1,\\ -D & \text{otherwise,} \end{cases}\\ \tilde{c}^k[j] &= \begin{cases} \mathrm{pre}_k(c[j]) & \text{if } \mathrm{pre}_{k+1}(c[j]) = 2 \cdot \mathrm{pre}_k(c[j]),\\ D & \text{otherwise.} \end{cases} \end{aligned}$$

Observe that if any of the conditions 5.2 – 5.4 is not satisfied, then the unrolled formula $\tilde{a}^k[i] + \tilde{b}^k[j] = \tilde{c}^k[i + j]$ contains at least one summand $D$ and thus cannot be satisfied. Otherwise, it reduces to the condition 5.5.

The inequality $a[i] + b[j] > c[i + j]$ holds for some $i, j$ iff one of the constructed instances of 3sumConv returns *true*. As the number of instances is $\mathcal{O}(\log W)$, the claim follows. The 3sumConv problem is subquadratically equivalent to 3SUM [124], which establishes a relationship between these two classes of subquadratic equivalence. □

## 5.6 Nondeterministic algorithm for 3SUM

Carmosino et al. [34, Lemma 5.8] presented an $\mathcal{O}(n^{1.5})$ nondeterministic algorithm for 3SUM, i.e., the running time depends only on the size of the input. However, in our application, we need a running time that is a function of the sizes of sets $A$,$B$ and $C$. In this section we analyze the running time of Carmosino et al. in regard to these parameters.

**Lemma 5.6.1.** *There is a nondeterministic algorithm for* 3SUM *with running time*
$\mathcal{O}(\sqrt{n_1 n_2 n_3}\log^2(W))$, *where* $n_1 = |A|$, $n_2 = |B|$, $n_3 = |C|$ *and* $W$ *is the maximum absolute value of integers in* $A \cup B \cup C$ *(we assume that* $n_1 + n_2 + n_2 \leq W$*).*

*Proof.* If there is a triple ($a \in A$, $b \in B$, $c \in C$) such that $a + b = c$, then we can nondeterministically guess it and verify it in $\mathcal{O}(1)$ time. To prove that there is no such triple, we nondeterministically guess the following:

1. a prime number $p \leq prime_{\sqrt{n_1 n_2 n_3}}$, where $prime_i$ denotes the $i$-th prime number,

2. an integer $t(p) \leq \sqrt{n_1 n_2 n_3}\log(3W)$, which is the number of solutions for sets
   ($A \pmod p$, $B \pmod p$, $C \pmod p$),

3. a set $S = \{(a_1, b_1, c_1), \ldots, (a_{t(p)}, b_{t(p)}, c_{t(p)})\}$, where $|S| = t(p)$ and each triple ($a_i \in A$, $b_i \in B$, $c_i \in C$) satisfies $a_1 + b_1 \equiv c_1 (\mathrm{mod}\, p)$.

To see that for each NO-instance there exists such a proof, consider the number of false positives, that is, tuples ($a \in A$, $b \in B$, $c \in C$, $p$), where $p$ is a prime. For each triple ($a \in A$, $b \in B$, $c \in C$), the value $|a + b - c|$ has at most $\log(3W)$ distinct prime divisors. Therefore, the number of false positives is bounded by $n_1 n_2 n_3 \log(3W)$. Since there are $\sqrt{n_1 n_2 n_3}$ candidates for $p$, we can choose one such that $t(p) \leq \sqrt{n_1 n_2 n_3}\log(3W)$.

To verify the proof, we need to verify whether $S$ contains no true solution and to compute the number of solutions for ($A \pmod p$, $B \pmod p$, $C \pmod p$). If it equals to $|S|$, then we are sure that all solutions for the instance modulo $p$ are indeed false positives for the original instance. Since the numbers are bounded by $p$, we can count the solutions using FFT in time $\mathcal{O}(p \log p) = \mathcal{O}(\sqrt{n_1 n_2 n_3}\log^2(W))$. □

## 5.7 Approximate Min-Plus Convolution

**Definition 5.7.1** (Approximate Min-Plus Convolution)**.** *In approximate Min-Plus Convolution problem we are given sequences* $A[0,\ldots,n-1]$, $B[0,\ldots,n-1]$ *of positive integers and approximation parameter* $0<\varepsilon<1$. *Let* $OPT[k] = \min_{0\le i\le k}(A[i]+B[k-i])$ *be the Min-Plus Convolution of* $A$ *and* $B$. *The task is to find a sequence* $C[0,\ldots,n-1]$ *such that* $\forall_i\ OPT[i]\le C[i]\le(1+\varepsilon)OPT[i]$

Backurs, Indyk, and Schmidt [18] described a $(1+\varepsilon)$-approximation algorithm for Min-Plus Convolution, that runs deterministically in time $\mathcal{O}(\frac{n}{\epsilon^2}\log n\log^2 W)$. In their paper [18] it is used as a building block to show a near-linear time approximation algorithm for Tree Sparsity. With the approximation algorithm for Min-Plus Convolution, they managed to solve Tree Sparsity approximately in $\widetilde{\mathcal{O}}(\frac{n}{\varepsilon^2})$ time, which in practical applications may be faster than solving this problem exactly in time $\widetilde{\mathcal{O}}(n^2)$.

In this Section we improve upon the $\widetilde{\mathcal{O}}(n/\varepsilon^2)$ approximation algorithm. for Min-Plus Convolution. Similar techniques have been exploited to obtain the $\widetilde{\mathcal{O}}(n^\omega/\varepsilon)$-time approximation for APSP [169] and they have found use in the approximate pattern matching over $l_\infty$ [109]. The basic idea is to propose a fast exact algorithm depending on $W$ (upper bound on the weights) and apply it after rounding weights into smaller space. Our result also applies to Max-Plus Convolution.

### 5.7.1 State of the art

In this Section we will shortly describe the $\widetilde{\mathcal{O}}(\frac{n}{\varepsilon^2})$ algorithm due to Backurs, Indyk, and Schmidt [18, Section C.2]. The basic idea is to round elements and then perform a fast convolution with FFT. For a sequence $D[0,\ldots,n-1]$ and integer $i$, we define a binary vector $\lambda(D,i)$:

$$\lambda(D,i)[k] := \begin{cases} 1 & \text{if } (1+\epsilon)^i \le D[k] \le (1+\epsilon)^{i+1}, \\ 0 & \text{otherwise} \end{cases}$$

The vector $\lambda(D,i)$ is nonzero in the bits that round the values of sequence $D$. Let $W$ be the largest value in both $A$ and $B$. Then for all pairs of indices $0\le i,j\le\log_{1+\varepsilon}W$ we define a vector $\lambda_{i,j}=\lambda(A,i)\oplus\lambda(B,j)$ ($\oplus$ stands for convolution of the sequences). Computation of a single vector $\lambda_{i,j}$ takes $\mathcal{O}(n\log n)$ time and there are $\mathcal{O}(\log^2_{1+\varepsilon}W)$ such vectors. Hence the total running time is $\mathcal{O}(n\log n\log^2_{1+\varepsilon}W)=\mathcal{O}(\frac{n}{\varepsilon^2}\log n\log^2 W)$.

Finally we iterate over all entries and output

$$C[k] = \min_{(\lambda_{i,j})_k=1} (1+\varepsilon)^{i+1} + (1+\varepsilon)^{j+1},$$

which gives us the approximate answer for all entries.

### 5.7.2 Exact $\widetilde{\mathcal{O}}(nW)$ algorithm

The Min-Plus Convolution admits a brute force $\mathcal{O}(n^2)$-algorithm. From the other hand, when all values in sequences are binary, then applying FFT and performing convolution yields an $\mathcal{O}(n \log n)$-algorithm. Our exact $\widetilde{\mathcal{O}}(nW)$ algorithm is an attempt to capture this trade-off. Note, that this algorithm is worse than a brute force whenever $W > n$ which is often the case. However, this algorithm turns out useful for approximation.

**Lemma 5.7.2.** *The Min-Plus Convolution* [Max-Plus Convolution ] *problem can be solved deterministically in* $\mathcal{O}(nW \log(nW))$ *time and* $\mathcal{O}(nW)$ *space.*

*Proof.* Given sequences $A[0, \ldots, n-1]$ and $B[0, \ldots, n-1]$ with values at most $W$, we transform them into binary sequences of length $2nW$. We encode every number in the natural unary manner. For $0 \leq i < n$, $1 \leq k \leq W$ we define:

$$\tilde{a}[2Wi + k] = \begin{cases} 0 & \text{if } A[i] \neq k \\ 1 & \text{if } A[i] = k \end{cases}$$

and similarly we define sequence $\tilde{B}$.

We compute convolution $\tilde{C} = \tilde{A} \oplus \tilde{B}$ using FFT in time $\mathcal{O}(nW \log n \log W)$. Since

$$\tilde{C}[2Wi + k] = \sum_{\substack{i_1+i_2=i \\ k_1+k_2=k}} \tilde{A}[2Wi_1 + k_1] \cdot \tilde{B}[2Wi_2 + k_2],$$

the first nonzero occurrence in the $i$-th block of length $2W$ encodes the value of the $i$-th element of the requested Min-Plus Convolution. If we are interested in computing Max-Plus Convolution, we should similarly seek for last nonzero value in each block.

The time complexity is dominated by performing convolution with FFT. As the additional space we need $\mathcal{O}(nW)$ bits for the transformed sequences. □

### 5.7.3 Approximation

We start with a lemma inspired by [169, Lemma 5.1] and [109, Lemma 1].

**Lemma 5.7.3** (Rounding Lemma)**.** *For natural numbers $x, y$ and positive $q, \varepsilon$ satisfying $q \leq x + y$ and $0 < \varepsilon < 1$ it holds:*

$$\begin{aligned} x + y &\leq \left(\left\lceil \tfrac{2x}{q\epsilon} \right\rceil + \left\lceil \tfrac{2y}{q\epsilon} \right\rceil\right)\tfrac{q\epsilon}{2} &< (x+y)(1+\epsilon), \\ (x+y)(1-\epsilon) &< \left(\left\lfloor \tfrac{2x}{q\epsilon} \right\rfloor + \left\lfloor \tfrac{2y}{q\epsilon} \right\rfloor\right)\tfrac{q\epsilon}{2} &\leq x + y. \end{aligned}$$

The proof of above Lemma is a special case of Lemmas 2.4.1 and 2.4.2 for $k = 2$

Now we are ready to give a scaling based approximation algorithm for Min-Plus Convolution.

**Lemma 5.7.4.** *Assume the Min-Plus Convolution* [Max-Plus Convolution ] *can be solved exactly in time $T(n, W)$. Then we can approximate Min-Plus Convolution* [Max-Plus Convolution ] *in time $\mathcal{O}((T(n, \frac{4}{\varepsilon}) + n) \log W)$.*

**Algorithm 6** ApproximateMinConv$(A, B)$. We use a simplified notation to transform all elements in the sequences $A[i]$ and $B[i]$.

```
1: Output[i] = ∞
2: for l = 2⌈log W⌉, ..., 0 do
3:     q := 2^l
4:     A'[i] = ⌈2A[i]/(qε)⌉
5:     if A'[i] > ⌈4/ε⌉ then
6:         A'[i] = ∞
7:     end if
8:     B'[i] = ⌈2B[i]/(qε)⌉
9:     if B'[i] > ⌈4/ε⌉ then
10:        B'[i] = ∞
11:    end if
12:    V = runExact(A', B')
13:    if V[i] < ∞ then
14:        Output[i] = V[i] · qε/2
15:    end if
16: end for
17: return Output[0, ..., n − 1]
```

*Proof.* The idea is based on [109, Section 6.2]. We focus on the variant with Min-Plus Convolution, however the proofs works alike for Max-Plus Convolution.

We iterate the *precision parameter* $q$ through $2W, W, \ldots, 4, 2, 1$. In each iteration we apply the transform from Lemma 5.7.3 ($x \to \left\lceil \frac{2x}{q\epsilon} \right\rceil$) to all elements in $A, B$, we set $\infty$ for each value exceeding $\left\lceil \frac{4}{\varepsilon} \right\rceil$, and launch the exact algorithm on such input. We multiply all finite elements in the returned array by $\frac{q\epsilon}{2}$ and store them in the output array $C$, possibly overwriting some elements.

Assume the correct value of $C[k]$ equals $A[i]+B[k-i]$. For some iteration we get the precision parameter $q$ such that $q \leq C[k] < 2q$. The rounded numbers $\left\lceil \frac{2A[i]}{q\epsilon} \right\rceil$, $\left\lceil \frac{2B[k-i]}{q\epsilon} \right\rceil$ are at most $\left\lceil \frac{4}{\varepsilon} \right\rceil$, so we will update the $k$-th index in the output array. On the other hand, the assumption of Lemma 5.7.3 is satisfied, therefore the generated value lies between $C[k]$ and $C[k](1+\varepsilon)$. In the following iterations, we will still have $q \leq C[k]$, therefore any further updates to the $k$-th index will remain valid.

The algorithm performs $\mathcal{O}(\log W)$ iterations and in each step we run the exact algorithm in time $T(n, \frac{4}{\epsilon})$, thanks to the pruning procedure. Transforming the sequences takes $\mathcal{O}(n)$ time in each step. □

**Theorem 5.7.5** (Apx for Min-Plus Convolution/Max-Plus Convolution)**.** *There is a deterministic algorithm for $(1+\varepsilon)$-approximate Min-Plus Convolution [Max-Plus Convolution ] running in $\mathcal{O}\left(\frac{n}{\varepsilon} \log\left(\frac{n}{\varepsilon}\right) \log W\right)$ time.*

*Proof.* From Lemma 5.7.2 the running time of exact algorithm is $T(n, W) = \mathcal{O}(nW \log n \log W)$. This quantity dominates the additive term $\mathcal{O}(n \log W)$. Hence by replacing each $W$ with $1/\varepsilon$ we get the claimed running time. □

## 5.8 Approximate Tree Sparsity

The approximation version of Tree Sparsity comes with two flavors: as a *head* approximation where we are supposed to maximize the weight of the solution, and as a *tail* approximation where we minimize the total weight of nodes that do not belong to the solution. Note that a constant approximation for one of the variants does not necessarily yield a constant approximation for the other one. Backurs, Indyk, and Schmidt [18] proposed an $\mathcal{O}\left(\frac{n}{\varepsilon^2} \cdot \log^{12} n \cdot \log^2 W\right)$ running time for $(1-\varepsilon)$-head approximation, and an $\mathcal{O}\left(\frac{n}{\varepsilon^3} \cdot \log^9 n \cdot \log^3 W\right)$ running time for $(1+\varepsilon)$-tail approximation.

In this section we improve the running times for both variants relying on the $\tilde{\mathcal{O}}\left(\frac{n}{\varepsilon}\right)$ algorithm for approximating Min-Plus Convolution and Max-Plus Convolution.

The following theorem, combined with our approximation for Min-Plus Convolution yields an $\mathcal{O}\left(\frac{n}{\varepsilon} \cdot \log(n/\varepsilon) \cdot \log^3 n \cdot \log W\right)$-time algorithm that

computes the maximal weights of rooted subtrees for each size $k = 1, \ldots, n$ with a relative error at most $\varepsilon$ in both head and tail variant.

**Theorem 5.8.1.** *If $(1+\varepsilon)$-approximate Min-Plus Convolution can be solved in time $T(n, W, \varepsilon)$, then $(1+\varepsilon)$-approximate Tree Sparsity can be solved in time $\mathcal{O}\Big(\Big(n + T(n, W, \varepsilon/\log^2 n)\Big)\log n\Big)$.*

*Proof.* We exploit the heavy-light decomposition introduced by Sleator and Tarjan [140]. This technique has been utilized by Backurs, Indyk, and Schmidt [18] in their work on Tree Sparsity approximation.

We construct a *spine* with a *head* $s_1$ at the root of the tree. We define $s_{i+1}$ to be the child of $s_i$ with the larger subtree (in case of draw we choose any child) and the last node in the spine is a leaf. The remaining children of nodes $s_i$ become heads for analogous spines so the whole tree gets covered. Observe that every path from a leaf to the root intersects at most $\log n$ spines because each spine transition doubles the subtree size.

At first we express the head variant in the convolutional paradigm. For a node $v$ with a subtree of size $m$ we define the sparsity vector

$$(x^v[0], x^v[1], \ldots, x^v[m])$$

of weights of the heaviest subtrees rooted at $v$ with fixed sizes. This vector equals the Max-Plus Convolution of the sparsity vectors for the children of $v$. We are going to compute sparsity vectors for all heads of spines in the tree recursively. Having this performed we can read the solution from a sparsity vector of the root. Let $(s_i)_{i=1}^{\ell}$ be a spine with a head $v$ and let $u^i$ indicate the sparsity vector for the child of $s_i$ being a head (i.e., the child with the smaller subtree). If $s_i$ has less than two children we treat $u^i$ as a vector $(0)$.

For an interval $[a, b] \subseteq [1, \ell]$ let $u^{a,b} = u^a \oplus^{\max} u^{a+1} \oplus^{\max} \cdots \oplus^{\max} u^b$ and $y^{a,b}[k]$ be the maximum weight of a subtree of size $k$ rooted at $s_a$ and not containing $s_{b+1}$. Let $c = \left\lfloor \frac{a+b}{2} \right\rfloor$. The $\oplus^{\max}$ operator is associative so $u^{a,b} = u^{a,c} \oplus^{\max} u^{c+1,b}$. To compute the second vector we consider two cases: whether the optimal subtree contains $s_{c+1}$ or not.

$$\begin{aligned} y^{a,b}[k] &= \max\left[y^{a,c}[k], \quad \sum_{i=a}^{c} x(s_i) + \max_{k_1+k_2=k-(c-a+1)} \Big(u^{a,c}[k_1] + y^{c+1,b}[k_2]\Big)\right] \\ &= \max\left[y^{a,c}[k], \quad \sum_{i=a}^{c} x(s_i) + \Big(u^{a,c} \oplus^{\max} y^{c+1,b}\Big)\Big[k-(c-a+1)\Big]\right] \end{aligned} \tag{5.6}$$

Using the presented formulas we reduce the problem of computing $x^v = y^{1,\ell}$ to subproblems for intervals $[1, \frac{\ell}{2}]$ and $[\frac{\ell}{2}+1, \ell]$ and results are merged

with two $(\max, +)$-convolutions. Proceeding further we obtain $\log \ell$ levels of recursion. Since there are $\mathcal{O}(\log n)$ spines on a path from a leaf to the root, the whole computation tree has $\mathcal{O}(\log^2 n)$ layers, each node being expressed as a pair of convolutions on vectors from its children. Each vertex of the graph occurs in at most $\log n$ convolutions so the sum of convolution sizes is $\mathcal{O}(n \log n)$.

In order to deal with the tail variant we consider a dual sparsity vector $(\overline{x}^v[0], \overline{x}^v[1], \ldots, \overline{x}^v[m])$, where $\overline{x}^v[i]$ stands for the total weight of the subtree rooted at $v$ minus $x^v[i]$. The dual sparsity vector of $v$ equals the Min-Plus Convolution of the vectors for the children of $v$. We can use an analog of equation (5.6) and also express the problem as a computation tree based on convolutions.

We take advantage of Theorem 5.7.5 to perform each convolution with a relative error $\delta$. The formula (5.6) contains an additive term $\sum_{i=a}^{c} x(s_i)$ but this can only decrease the relative error. The cumulative relative error is bounded by $(1-\delta)^{\log^2 n}$ for head approximation and $(1+\delta)^{\log^2 n}$ for tail approximation, therefore setting $\delta = \Theta(\varepsilon / \log^2 n)$ guarantees that the sparsity vector for the root is burdened with relative error at most $\varepsilon$.

The sum of running times for all convolutions is $\mathcal{O}\Big(T(n, W, \delta) \log n\Big)$, what gives the postulated running time for the whole algorithm. In order to retrieve the solution for a given $k$, we need to find the pair of indices that produced the value of the $k$-th index of the last convolution. Then we proceed recursively and traverse back the computation tree. Since finding $\arg\max$ and $\arg\min$ can be performed in linear time, the total time of analyzing all convolutions is $\mathcal{O}(n \log n)$. □

## 5.9 $\widetilde{\mathcal{O}}(n + 1/\varepsilon)$ approximation algorithm for 3SUM

In this section we will show an $\widetilde{\mathcal{O}}(n+1/\varepsilon)$ approximation algorithm for 3SUM and prove accompanying lower bound under a reasonable assumption.

**Definition 5.9.1** ($k$-SUM)**.** *In $k$-SUM problem, one is given sets $A_1$,$A_2$,..., $A_{k-1}$, $S$, each with cardinality at most $n$. The task is to decide if there is a tuple $(a_1, \ldots, a_{k-1}, s) \in A_1 \times \ldots \times A_{k-1} \times S$ such that $a_1 + \ldots + a_{k-1} = s$.*

The 3SUM problem is a special case of $k$-SUM for $k = 3$. The 3SUM is one of the most notorious problems with a quadratic running time and has been widely accepted as a hardness assumption (see [159] for overview). The fastest known algorithm for 3SUM is slightly subquadratic: Patrascu [124]

gave an $\mathcal{O}(n^2(\log\log n/\log n)^{2/3})$-time deterministic algorithm and then independently Gajentaan and Overmars [67] and Gold and Sharir [79] improved this result by presenting an $\mathcal{O}(n^2\log\log n/\log n)$-time algorithm.

The approximation variant for 3SUM was considered by Gfeller [77] who showed a deterministic $\widetilde{\mathcal{O}}(\frac{n}{\varepsilon})$ algorithm as a byproduct of finding longest approximate periodic patterns. If we are not interested in exact solution, the Gfeller [77] algorithm is polynomially faster than the best exact algorithm for 3SUM. In this section we show how to solve 3SUM approximately in time $\widetilde{\mathcal{O}}(n+1/\varepsilon)$ time and prove this tight up to the polylogarithmic factors.

**Definition 5.9.2** (Approximate 3SUM ([77]))**.** *In the approximate 3SUM we are given three sets $A$, $B$, $C$ of positive integers, each with cardinality at most $n$. The algorithm that solves approximate 3SUM, either:*

- *it outputs a triple $(a,b,c)\in A\times B\times C$ with $a+b\in[c/(1+\varepsilon),\, c(1+\varepsilon)]$, or*
- *concludes that no triple $(a,b,c)\in A\times B\times C$ with $a+b=c$ exists.*

This definition generalizes to $k$-SUM, however we are unaware about any previous works on approximate $k$-SUM.

### 5.9.1 Faster approximation algorithm for 3SUM

In this section we present an $\widetilde{\mathcal{O}}(n+1/\varepsilon)$-time approximation scheme for 3SUM problem. We use a technique from Section 5.7, where we gave the fast approximation algorithm for Min-Plus Convolution. As previously, we start with a fast $\widetilde{\mathcal{O}}(n+W)$ exact algorithm and then utilize rounding to get an approximation algorithm. In the Section 5.10 we will show a conditional optimality of this result.

#### Exact $\widetilde{\mathcal{O}}(n+W)$ algorithm for 3SUM

Let $W$ denote the upper bound on the integers in the sets $A$,$B$ and $C$. The exact $\widetilde{\mathcal{O}}(n+W)$-time algorithm for 3SUM is already well known [39, 46]. In here we will place the proof for completeness. For formal reasons we need to take care of the special symbol $\infty$. What is more, we will generalize this result to $k$-SUM.

**Theorem 5.9.3** (Based on [39, 46])**.** *The $k$-SUM can be solved deterministically in $\widetilde{\mathcal{O}}(kn+kW\log W)$ time and $\widetilde{\mathcal{O}}(kn+W)$ space.*

*Proof.* We will encode the numbers in the sets as binary arrays of size $\mathcal{O}(W)$ and iteratively perform fast convolution using FFT. Because we will use only $\mathcal{O}(1)$ tables at once, the space complexity will not depend on $k$. At the end we will need to check if any entry in the final array is in $S$.

**Encoding:** We iterate for every set $A_1, \ldots, A_{k-1}$ and for $l$-th iteration encode it as a binary vector $V$ of length $W+1$, such that:

$$V_l[i] = \begin{cases} 1 & \text{iff } t \in A_l \\ 0 & \text{otherwise} \end{cases}$$

to save space we will use only one $V_l$ vector at the time. The encoding can be done in $\mathcal{O}(n+W)$ time. If the special symbol $\infty \in A_l$ appears then we simply discard it.

**FFT:** We want to perform a convolution with FFT on all vectors $V_l$. We do it one at a time and discard all elements larger than $W$. Let $U_l$ be the result of up to $l$-th iteration. Let the polynomial $U_l$ be

$$U_l(x) = \sum_{(a_1,\ldots,a_l)\in(A_1\times\ldots\times A_l)} x^{a_1+\ldots a_l}.$$

If we multiply it by the polynomial $V_{l+1} = \sum_{a_{l+1}\in A_{l+1}} x^{a_{l+1}}$, we get

$$U_{l+1}(x) = \sum_{(a_1,\ldots,a_{l+1})\in(A_1\times\ldots\times A_{l+1})} x^{a_1+\ldots a_{l+1}}.$$

In the end of that proces, we get the vector $V_{k-1}$ that encodes all the sums of elements in subsets truncated up to $W$ order.Then, we get the binary vector for $S$ and compare it with the resulting vector $V_{k-1}$.

**Time and Space** We did $k$ iterations. In each of them we transformed a set into a vector in time $\mathcal{O}(n)$. The fast convolution works in $\mathcal{O}(T \log T)$ by using FFT. Hence, the running time is $\mathcal{O}(kn + kW \log W)$. Algorithm needs $\mathcal{O}(nk)$ space to encode input and $\mathcal{O}(W)$ space to store binary vectors. □

### Approximation algorithm

Next we will use an exact algorithm to propose the fast approximation. We will use the same reasoning as in Section 5.7.5.

**Lemma 5.9.4.** *Assume the $k$-SUM can be solved exactly in $T(n,k,W)$ time. Then approximate $k$-SUM can be solved in $\mathcal{O}((T(n,k,k/\varepsilon)+nk)\log W)$ time.*

**Algorithm 7** ApproximateKSum$(a_1, a_2, \ldots, a_{k-1}, s, \varepsilon)$. We use a shorten notation to transform all elements in the sequences $a_l[i]$ and $s[i]$.

Output$[i] = \infty$
**for** $l = 2\lceil \log W \rceil, \ldots, 0$ **do**
    $q := 2^l$
    **for** $l = 1 \ldots k-1$ **do**
        $a'_l[i] = \left\lceil \frac{k a_l[i]}{q\varepsilon} \right\rceil$
        **if** $a'_l[i] > \lceil 4k/\varepsilon \rceil$ **then**
            $a'_l[i] = \infty$
        **end if**
    **end for**
    $s'[i] = \left\lceil \frac{k s[i]}{q\varepsilon} \right\rceil$
    **if** $s'[i] > \lceil 4k/\varepsilon \rceil$ **then**
        $s'[i] = \infty$
    **end if**
    **if** runExactKsum$(a'_1, \ldots, a'_{k-1}, s')$ **then**
        **return** True
    **end if**
**end for**
**return** False

*Proof of Lemma 5.9.4.* We follow the Proof of Lemma 5.7.4. Assume, that there is some number $s$, such that there exists a tuple $(a_1, a_2, \ldots, a_{k-1}) \in A_1 \times \ldots A_{k-1}$, that $s < \sum_{i=1}^{k} a_i < s(1+\varepsilon)$. Observe, that in Algorithm 7 we iterate over precision parameter $q$, hence there exists some $q$, such that $q \leq s < 2q$. From Lemma 2.4.1 we know, that we can round the numbers $a'_i = \left\lceil \frac{ka_i}{q\varepsilon} \right\rceil$ and then their sum is

$$\sum_{i=1}^{k} a_i \leq \sum_{i=1}^{k} \left\lceil \frac{ka_i}{q\varepsilon} \right\rceil < (1+\varepsilon) \sum_{i=1}^{k} a_i.$$

So if there is some number $s \in S$, then ApproximateKSum algorithm finds a tuple, that sums up to $s' \in [s, (1+\varepsilon)s]$. Analogously, we can use Lemma 2.4.2 and round up numbers to find a tuple, that sums up to $s' \in [(1-\varepsilon)s, s]$.

For completness, assume that for all $s \in S$ no tuple sums up to $s' \in [(1-\varepsilon)s, (1+\varepsilon)s]$. We need to prove that in such a case, ApproximateKSum returns NO. However, for such $s'$ there always exists a precision parameter $q$, that $q \leq s' < 2q$. Then rounding the numbers according to Lemma 2.4.1 and Lemma 2.4.2 guarantees $(1 \pm \varepsilon)$ error. Hence, numbers after rounding cannot sum to $\left\lceil \frac{ks}{q\varepsilon} \right\rceil$. So, if for all $s \in S$ no tuple sums up to $[(1-\varepsilon)s, (1+\varepsilon)s]$ then our ApproximateKSum will return NO.

In the Definition 5.9.2 we need to return approximation of the form OPT$/(1+\varepsilon) \leq$ OPT' $\leq$ OPT$(1+\varepsilon)$ but note that $\frac{1}{1+\varepsilon} \approx 1-\varepsilon$ so we can take care of it by mulitplying $\varepsilon$ by a constant. □

At the end we need to connect the exact algorithm from Lemma 5.9.3 and the reduction from Lemma 5.9.4.

**Theorem 5.9.5.** *There is a deterministic algorithm for $(1+\varepsilon)$-approximate $k$-SUM running in $\mathcal{O}(nk \log W + \frac{k^2}{\varepsilon} \log \frac{k}{\varepsilon} \log W)$ time.*

*Proof.* From Lemma 5.9.3 the running time of $k$-SUM is $T(n,k,W) = \mathcal{O}(nk + kW \log W)$. Applying this running time to the reduction in Lemma 5.9.4 results in the claimed running time, because the $\mathcal{O}(nk)$ term is dominated by $\mathcal{O}(nk \log W)$ term in the reduction. □

To get an approximate algorithm for 3SUM we set $k = 3$.

**Corollary 5.9.6.** *The approximate 3SUM can be solved deterministically in $\mathcal{O}((n + \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}) \log W))$ time.*

## 5.10 Conditional lower bounds for approximate Knapsack-type problems

Abboud et al. [3] showed that assuming SETH there can be no $\mathcal{O}(t^{1-\delta}\text{poly}(n))$ algorithm for Subset Sum (Cygan et al. [47] obtained the same lower bound before but assuming the SetCover conjecture).

**Theorem 5.10.1** (Conditional Lower Bound for approximate Subset Sum)**.** *For any constant $\delta > 0$, a weak $(1-\varepsilon)$-approximation for Subset Sum with running time $\mathcal{O}\left(\text{poly}(n)\left(\frac{1}{\varepsilon}\right)^{1-\delta}\right)$ would refute SETH and SetCover conjecture.*

*Proof.* We set $\varepsilon = 2/t$ and obtain an algorithm that solves an exact Subset Sum, because all numbers are integers and the absolute error is at most $1/2$. The running time is $\mathcal{O}(t^{1-\delta}\text{poly}(n))$. This refutes SETH by the reduction in [3] and the SetCover conjecture by the reduction in [47]. □

### 5.10.1 Conditional lower bound for approximate 3SUM

We have shown an approximate algorithm for 3SUM running in $\widetilde{\mathcal{O}}(n + 1/\varepsilon)$ time. Is this the best we can hope for? Perhaps one could imagine an $\widetilde{\mathcal{O}}(n+1/\sqrt{\varepsilon})$ time algorithm. In this subsection we rule out such a possibility and prove the optimality of Theorem 5.9.6.

To show the conditional lower bound we will assume the hardness of the exact 3SUM. The 3SUM conjecture says, that the $\widetilde{\mathcal{O}}(n^2)$ algorithm is essentially the best we can hope for up to subpolynomial factors.

**Conjecture 5.10.2** (3SUM conjecture [159])**.** *In the Word RAM model with $\mathcal{O}(\log n)$ bit words, any algorithm requires $\Omega(n^{2-o(1)})$ time in expectation to determine whether given set $S \subset \{-n^{3+o(1)}, \ldots, n^{3+o(1)}\}$ of size $n$ contains three distinct elements $a, b, c$ such that $a + b = c$.*

Solving 3SUM with polynomially bounded numbers can be reduced to solving it with the upper bound $W = \mathcal{O}(n^{3+o(1)})$ [20]. 3SUM can be solved in subquadratic time when $W = \mathcal{O}(n^{2-\delta})$ via FFT. Hsu and Umans [88] have considered it as a yet another hardness assumption.

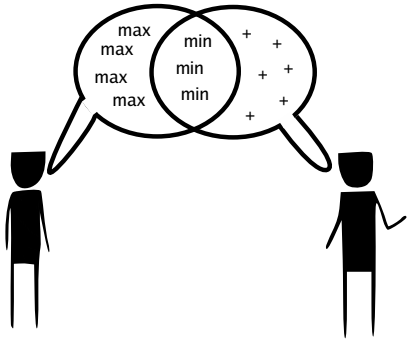
**Conjecture 5.10.3** (Strong-3SUM conjecture [88])**.** *3SUM on a set of $n$ integers in the domain of $\{-n^2, \ldots, n^2\}$ requires time $\Omega(n^{2-o(1)})$.*

**Theorem 5.10.4.** *Assuming the Strong-3SUM conjecture, there is no $\widetilde{\mathcal{O}}(n + 1/\varepsilon^{1-\delta})$ algorithm for $(1+\varepsilon)$-approximate 3SUM, for any constant $\delta > 0$.*

*Proof.* Consider the exact variant of 3SUM within the domain $\{-n^2, \ldots, n^2\}$. We can assume that the numbers are divided into sets $A, B, C$ and we can restrict ourselves to triples $a \in A$, $b \in B$, $c \in C$ [20]. We add $n^2 + 1$ to all numbers in $A \cup B$ and likewise $2n^2 + 2$ to all numbers in $C$ to obtain an equivalent instance with all input numbers greater than 0 and $W = \mathcal{O}(n^2)$.

Suppose, that for some small $\delta > 0$ the approximate 3SUM admits an $\widetilde{\mathcal{O}}(n + 1/\varepsilon^{1-\delta})$-algorithm. We can use it to solve the constructed instance exactly by setting $\varepsilon = \frac{1}{2W} = \Omega(n^{\frac{1}{2}})$. The running time of the exact algorithm is strongly subquadratic, namely $\widetilde{\mathcal{O}}(n+1/\varepsilon^{1-\delta}) = \widetilde{\mathcal{O}}(n^{2-2\delta})$. This contradicts the Strong-3SUM conjecture.

□




# Chapter 6

# Approximate (min,+) is equivalent to (min,max)

In this Chapter, we prove the equivalence of computing the approximate Min-Plus Product of two $n \times n$ matrices with arbitrary values and Min-Max Product. This yields a first subcubic strongly polynomial approximation for the All-Pairs Shortest Path problem (APSP). The algorithm runs in time $\mathcal{O}(n^{\frac{\omega+3}{2}}\varepsilon^{-1}\text{polylog}(n,\varepsilon)) \leq \widetilde{\mathcal{O}}(\frac{n^{2.69}}{\varepsilon})$ on word RAM. In comparison, a scaling-based approximation scheme for APSP runs in $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon}\log W)$ time (where $W$ is the upper bound on the weight) and is not strongly polynomial.

The result is rather unexpected. For applications of APSP (i.e., diameter, radius, minimum weight cycle, etc.) we show that the known algorithms can be transformed into a strongly polynomial relatively easily. Additionally, for undirected APSP we show that contracting edges and amortized analysis also give the power to obtain a $\widetilde{\mathcal{O}}(n^\omega/\varepsilon)$ strongly polynomial time. However, for APSP we cannot do such operations. In fact, this makes sense, as we prove that approximate APSP is at least as hard as exact Min-Max Product for which no $\mathcal{O}(n^\omega)$ algorithm is known. This can be seen as a conditional lower bound, although not based on a standard hypothesis. So we do not hope to get strongly polynomial $\mathcal{O}(n^\omega)$, but can break the cubic barrier? Yes we can!

**Outline:** Our main technical contribution is the following Sum-to-Max-Covering, which yields a framework for reducing approximate problems over the (min,+)-semiring to exact or approximate problems over the (min,max)-semiring. The most intriguing results of this thesis (the approximation scheme for directed APSP as well as the equivalence with Min-Max Product) are essentially immediate consequences of Sum-to-Max-Covering, see Sections 6.1 and 6.2.

**Theorem 6.0.1** (Sum-to-Max-Covering)**.** *Given vectors $A, B \in \mathbb{R}^n_+$ and $\varepsilon > 0$, in linear time in the output size we can compute vectors $A^{(1)}, \ldots, A^{(s)}$, $B^{(1)}, \ldots, B^{(s)} \in \mathbb{R}^n_+$ with $s = \mathcal{O}((\frac{1}{\varepsilon} + \log n) \log \frac{1}{\varepsilon})$ such that for all $i, j \in [n]$:*

$$A[i] + B[j] \leq \min_{\ell \in [s]} \max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \leq (1+\varepsilon)(A[i] + B[j]).$$

There are two main issues that make the proof of this statement non-trivial.

For *close pairs* $i, j$, meaning $\frac{A[i]}{B[j]} \in [\varepsilon, \frac{1}{\varepsilon}]$, the sum $A[i] + B[j]$ and the maximum $\max\{A[i], B[j]\}$ differ significantly. It is thus necessary to change the values of the vectors $A, B$. Roughly speaking, we handle this issue by splitting $A$ into vectors $A^{(\ell)}$ such that all entries $A^{(\ell)}[i], A^{(\ell)}[i']$ differ by either less than a factor $1 + \varepsilon$ or by more than a factor $\text{poly}(1/\varepsilon)$. Then we can choose $B^{(\ell)}$ such that $B^{(\ell)}[j]$ is approximately $A[i] + B[j]$ for all close pairs $i, j$. This ensures that for close pairs $\max\{A^{(\ell)}[i], B^{(\ell)}[j]\}$ is approximately $A[i] + B[j]$. For details see *Close Covering* (Lemma 6.3.2).

For the *distant pairs* $i, j$, with $\frac{A[i]}{B[j]} \notin [\varepsilon, \frac{1}{\varepsilon}]$, the sum $A[i] + B[j]$ and the maximum $\max\{A[i], B[j]\}$ differ by less than a factor $1+\varepsilon$, so we do not have to change any values. However, we need to remove some entries (by setting them to $\infty$) in order to not interfere with close pairs. We show how to cover all distant pairs but no too-close pairs, via a recursive splitting into $\log n$ levels of chunks and treating boundaries between chunks by introducing several shifts of restricted areas. For details see *Distant Covering* (Lemma 6.3.3).

We present our approximation scheme for APSP in Section 6.1 and the equivalence with Min-Max Product in Section 6.2. The main technical result, Max-to-Sum-Covering, is proved in Section 6.3. In Section 6.4 we discuss Undirected APSP, and in Section 6.5 we discuss certain graph characteristics. Finally, in Section 6.6 we present approximation scheme for Min-Plus Convolution and prove the equivalence with Min-Max Convolution.

The preliminary version of results in this Chapter was presented at *Symposium on Theory of Computing* (STOC 2019) [29].

## 6.1 Strongly polynomial approximation for directed APSP

We present a strongly polynomial $(1+\varepsilon)$-approximation algorithm for APSP with running time $\widetilde{\mathcal{O}}(n^{\frac{\omega+3}{2}}\varepsilon^{-1})$, proving Theorem 1.4.3. To this end, we first recall the reduction from approximate APSP to approximate Min-Plus Product from [170] (see Theorem 6.1.1). Then we observe that Sum-To-Max-Covering yields a reduction from approximate Min-Plus Product to Min-Max

Product. Using the known $\widetilde{\mathcal{O}}(n^{\frac{\omega+3}{2}})$-time algorithm for the latter shows the result (see Theorem 6.1.2).

**Theorem 6.1.1** (Implicit in [170])**.** *If $(1+\varepsilon)$-Approximate Min-Plus Product can be solved in time $T(n,\varepsilon)$, then $(1+\varepsilon)$-Approximate APSP can be solved in time $\mathcal{O}\Big(T(n,\varepsilon/\log n)\cdot\log n\Big)$.*

*Proof.* For the sake of completeness, we repeat the argument of Zwick [170, Theorem 8.1]. Let $A$ be the adjacency matrix of a given edge-weighted directed graph $G$, i.e., if there is an edge $(i,j)\in E$ of weight $w(i,j)$ then $A[i,j]=w(i,j)$, and $A[i,j]=\infty$ otherwise. We also add self-loops of weight 0, i.e, we set $A[i,i]=0$ for all $i\in[n]$. Given $\varepsilon>0$, we set $\varepsilon':=\ln(1+\varepsilon)/\lceil\log n\rceil$ (where ln is the natural logarithm and log is base 2). We will perform $\lceil\log n\rceil$ iterations of repeated squaring. In each iteration, we execute $(1+\varepsilon')$-Approximate Min-Plus Product on the current matrix $A$ with itself, i.e., we square the current matrix $A$. An easy inductive proof shows that after $r$ iterations each entry $A[i,j]$ is bounded from below by the distance from $i$ to $j$ in $G$, and bounded from above by $(1+\varepsilon')^r$ times the length of the shortest $2^r$-hop path from $i$ to $j$. Since any shortest path uses at most $n$ edges, after $\lceil\log n\rceil$ iterations each entry $A[i,j]$ is an approximation of the distance from $i$ to $j$ in $G$, by a multiplicative factor of

$$(1+\varepsilon')^{\lceil\log n\rceil}=\left(1+\frac{\ln(1+\varepsilon)}{\lceil\log n\rceil}\right)^{\lceil\log n\rceil}\leq 1+\varepsilon.$$

The direct running time of the reduction is $\mathcal{O}(n^2\log n)$ and there are $\mathcal{O}(\log n)$ calls to $(1+\varepsilon')$-Approximate Min-Plus Product with $\varepsilon'=\Theta(\frac{\varepsilon}{\log n})$. □

**Theorem 6.1.2.** *$(1+\varepsilon)$-Approximate Min-Plus Product can be solved in time $\widetilde{\mathcal{O}}(n^{\frac{\omega+3}{2}}\varepsilon^{-1})$.*

*Proof.* We use Sum-To-Max-Covering to reduce approximate Min-Plus Product to exact Min-Max Product and then use a known algorithm for the latter; the pseudocode is shown in Algorithm 8.

Consider input matrices $A,B\in\mathbb{R}_+^{n\times n}$ on which we want to compute $C\in\mathbb{R}_+^{n\times n}$ with $C[i,j]=\min_{k\in[n]}\{A[i,k]+B[k,j]\}$ for all $i,j\in[n]$. We view the matrices $A,B$ as vectors in $\mathbb{R}_+^{n^2}$, in order to apply Sum-To-Max-Covering (Lemma 6.3.1). This yields vectors $A^{(1)},\ldots,A^{(s)}$, $B^{(1)},\ldots,B^{(s)}\in\mathbb{R}_+^{n^2}$, which we re-interpret as matrices in $\mathbb{R}_+^{n\times n}$. We compute the Min-Max Product of every layer $A^{(\ell)},B^{(\ell)}$ and return the entry-wise minimum of the results, see Algorithm 8. (Note that we can replace the entries of $A^{(\ell)},B^{(\ell)}$ by their ranks before computing the Min-Max Product and then infer the actual result —

this is necessary since our input format for approximate Min-Plus Product is floating-point, but for Min-Max Product our input format is standard bit representation.)

**Algorithm 8** ApproximateMinProd$(A, B, \varepsilon)$.

1: $\{(A^{(1)}, B^{(1)}), \ldots, (A^{(s)}, B^{(s)})\} = \text{SumToMaxCovering}(A, B, \varepsilon)$
2: $C^{(\ell)} := \text{MinMaxProd}(A^{(\ell)}, B^{(\ell)})$ for all $\ell \in [s]$
3: $\tilde{C}[i,j] := \min_{\ell \in [s]} C^{(\ell)}[i,j]$ for all $i, j \in [n]$
4: **return** $\tilde{C}$

Let us prove that the output matrix $\tilde{C}$ is a $(1+\varepsilon)$-approximation of $C$. Sum-To-Max-Covering yields that for any $i, j, k$ we have

$$A[i,k] + B[k,j] \leq \min_{\ell \in [s]} \max\{A^{(\ell)}[i,k], B^{(\ell)}[k,j]\} \leq (1+\varepsilon)(A[i,k] + B[k,j]).$$

In particular, since

$$C[i,j] = \min_{\ell \in [s]} C^{(\ell)}[i,j] = \min_{\ell \in [s]} \min_{k \in [n]} \max\{A^{(\ell)}[i,k], B^{(\ell)}[k,j]\},$$

and $C[i,j] = \min_{k \in [n]}(A[i,k] + B[k,j])$, we obtain

$$C[i,j] \leq \tilde{C}[i,j] \leq (1+\varepsilon)C[i,j].$$

Sum-To-Max-Covering runs in time $\widetilde{\mathcal{O}}(n^2/\varepsilon)$. Computing $s$ times the Min-Max Product runs in time $\widetilde{\mathcal{O}}(sn^{\frac{\omega+3}{2}})$. We conclude the proof by noting that Sum-To-Max-Covering yields $s = \mathcal{O}(\frac{1}{\varepsilon}\operatorname{polylog}(n/\varepsilon))$. □

Combining Theorems 6.1.1 and 6.1.2 yields a $(1+\varepsilon)$-approximation for APSP in time $\widetilde{\mathcal{O}}(n^{\frac{\omega+3}{2}}\varepsilon^{-1})$.

## 6.2 Equivalence of approximate APSP and Min-Max Product

We next prove our equivalence of approximating APSP, exactly computing the Min-Max Product, and other problems. The theorem is restated here for convenience.

**Theorem 1.4.4.** *For any $c \geq 2$, if one of the following statements is true, then all are:*

- $(1+\varepsilon)$*-Approximate Directed APSP can be solved in strongly polynomial time* $\widetilde{\mathcal{O}}(n^c/\operatorname{poly}(\varepsilon))$*,*

- *$(1+\varepsilon)$-Approximate Min-Plus Product can be solved in strongly polynomial time $\widetilde{\mathcal{O}}(n^c/\mathrm{poly}(\varepsilon))$,*
- *exact Min-Max Product can be solved in strongly polynomial time $\widetilde{\mathcal{O}}(n^c)$,*
- *exact All-Pairs Bottleneck Path can be solved in strongly polynomial time $\widetilde{\mathcal{O}}(n^c)$.*

*Proof.* Equivalence of $(1+\varepsilon)$-Approximate APSP and $(1+\varepsilon)$-Approximate Min-Plus Product is essentially known. One direction is given by Theorem 6.1.1. For the other direction, given matrices $A, B$ we build a 3-layered graph, with edge weights between the first two layers as in $A$, edge weights between the last two layers as in $B$, and all edges directed from left to right. Then we observe that the pairwise distances between the first and third layers are in one-to-one correspondence to Min-Plus Product on $A, B$, also in an approximate setting.

Equivalence of Min-Max Product and All-Pairs Bottleneck Path is folklore (see, e.g., [52]). Both directions of this equivalence work exactly as for (approximate) Min-Plus Product vs. APSP.

Our main contribution is the equivalence of $(1+\varepsilon)$-Approximate Min-Plus Product and exact Min-Max Product. Observe that if Min-Max Product can be solved in time $T(n)$ then the algorithm from Theorem 6.1.2 runs in time $\widetilde{\mathcal{O}}(T(n)/\varepsilon)$.

It remains to show a reduction from Min-Max Product to $(1+\varepsilon)$-Approximate Min-Plus Product. Fix any constant $\varepsilon > 0$. Given matrices $A, B$, denote their Min-Max Product by $C$. Let $r$ be the value of $4(1+\varepsilon)^2$ rounded up to the next power of 2, and consider the matrices $A', B'$ with $A'[i,j] := r^{A[i,j]}$ and $B'[i,j] := r^{B[i,j]}$. (Recall that the input $A, B$ for Min-Max Product is in standard bit representation, so in constant time we can compute $r^{A[i,j]}$ in floating-point representation, by writing $A[i,j] \cdot \log r$ into the exponent.) Let $C'$ be the result of $(1+\varepsilon)$-Approximate Min-Plus Product on $A', B'$.

**Claim 6.2.1.** *We have $r^{C[i,j]} \le C'[i,j] \le r^{C[i,j]+1/2}$ for all $i, j$.*

Using this claim, we can infer $C$ from $C'$ by computing $C[i,j] = \lfloor \log_r C'[i,j] \rfloor$ (i.e., we simply read the most significant bits of the exponent of the floating-point number $C'[i,j]$). If $(1+\varepsilon)$-Approximate Min-Plus Product can be solved in time $T(n)$ (recall that $\varepsilon$ is fixed), then this yields an algorithm for Min-Max Product running in time $\widetilde{\mathcal{O}}(T(n))$. □

*Proof of Claim 6.2.1.* We will use $\min_k(A'[i,k] + B'[k,j]) \le C'[i,j] \le (1+\varepsilon)\min_k(A'[i,k] + B'[k,j])$ for all $i, j \in [n]$. For any $i, j$ there exists $k$ with

$C[i,j] = \max\{A[i,k], B[k,j]\}$. Hence,

$$\begin{aligned} C'[i,j] &\leq (1+\varepsilon)(A'[i,k] + B'[k,j]) = (1+\varepsilon)(r^{A[i,k]} + r^{B[k,j]}) \\ &\leq 2(1+\varepsilon) r^{\max\{A[i,k],B[k,j]\}} = 2(1+\varepsilon) r^{C[i,j]}, \end{aligned}$$

and by $r \geq 4(1+\varepsilon)^2$ we obtain $C'[i,j] \leq r^{C[i,j]+1/2}$. Moreover, for any $i,j$ there exists $k$ with $C'[i,j] \geq A'[i,k] + B'[k,j]$. We thus obtain

$$C'[i,j] \geq A'[i,k] + B'[k,j] = r^{A[i,k]} + r^{B[k,j]} \geq r^{\max\{A[i,k],B[k,j]\}} \geq r^{C[i,j]}. \quad \square$$

We remark that for scaling algorithms this proof shows an equivalence of the $\widetilde{\mathcal{O}}(Wn^\omega)$-time exact algorithm for Min-Max Product and the $\widetilde{\mathcal{O}}(\frac{n^\omega}{\mathrm{poly}(\varepsilon)} \log W)$-time approximation scheme for Min-Plus Product.

## 6.3 Sum-To-Max-Covering

In this section, we prove the main technical result of this Chapter, which we slightly reformulate here.

**Theorem 6.3.1** (Sum-to-Max-Covering, Reformulated)**.** *Given vectors $A, B \in \mathbb{R}_+^n$ and a parameter $\varepsilon > 0$, there are vectors $A^{(1)}, \ldots, A^{(s)}, B^{(1)}, \ldots, B^{(s)} \in \mathbb{R}_+^n$ with $s = \mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon} + \log n \log\frac{1}{\varepsilon})$ and:*

(i) *for all $i,j \in [n]$ and all $\ell \in [s]$:*

$$\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \geq A[i] + B[j], \text{and}$$

(ii) *for all $i,j \in [n]$ there exists $\ell \in [s]$:*

$$\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \leq (1+\varepsilon)(A[i] + B[j])$$

.

*We can compute such vectors $A^{(1)}, \ldots, A^{(s)}, B^{(1)}, \ldots, B^{(s)}$ in time $\mathcal{O}(\frac{n}{\varepsilon}\log\frac{1}{\varepsilon} + n\log n\log\frac{1}{\varepsilon})$.*

We split the construction into two parts, covering the pairs $i,j$ with $\frac{A[i]}{B[j]} \in [\varepsilon, 1/\varepsilon]$ (Close Covering Lemma, Section 6.3.1) and covering the remaining pairs (Distant Covering Lemma, Section 6.3.2). We show how to combine both cases in Section 6.3.3.

### 6.3.1 Close Covering

We first want to cover all pairs $i, j$ with $\frac{A[i]}{B[j]} \in [\varepsilon, 1/\varepsilon]$. To get an intuition, let $d \in \mathbb{Z}$ and consider only the entries $A[i]$ in the range $[(1+\varepsilon)^{d-1}, (1+\varepsilon)^d)$. Remove all other entries of $A$ by setting them to $\infty$, obtaining a vector $A'$. Since we consider the close case, we are only interested in entries $B[j]$ that differ by at most a factor $1/\varepsilon$ from $A[i]$, so consider the entries $B[j]$ in the range $[\varepsilon(1+\varepsilon)^{d-1}, \frac{1}{\varepsilon}(1+\varepsilon)^d)$. Add $(1+\varepsilon)^d$ to all such entries $B[j]$ and remove all other entries of $B$ by setting them to $\infty$, obtaining a vector $B'$. Then for the considered entries we have $\max\{A'[i], B'[j]\} = B'[j] = B[j] + (1+\varepsilon)^d$, which is between $A[i]+B[j]$ and $(1+\varepsilon)(A[i]+B[j])$. This covers all considered pairs in the sense of Max-to-Sum-Covering.

However, naively we would need to repeat this construction for too many values of $d$. The main observation of our construction is that we can perform this construction in parallel for all values $d \in D = \{s, 2s, 3s, \ldots\}$. That is, we only remove an entry of $A$ if it is irrelevant for *all* $d \in D$, and similarly for the entries of $B$. For a sufficiently large integer $s = \Theta(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$, it turns out that the considered entries for different $d$'s do not interfere. Performing this construction for all shifts $D+1, D+2, \ldots, D+s$ covers all close pairs. See Figure 6.1 for an illustration.

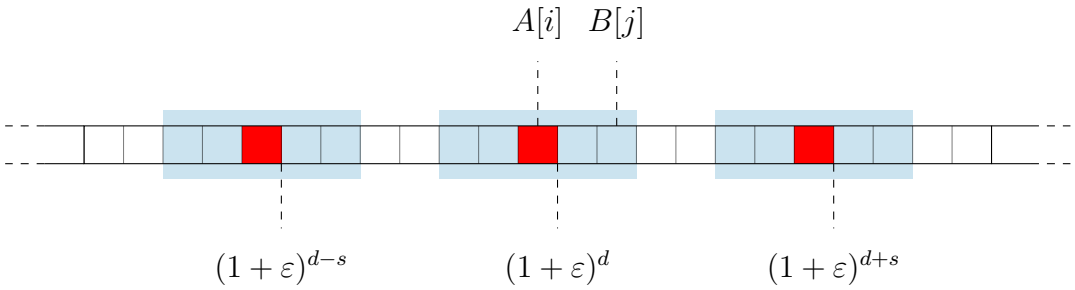



Figure 6.1: An illustration for Algorithm 9. Shown is the positive real line in log-scale. Entries of $A$ that lie outside the red/dark-shaded areas are set to $\infty$. Entries of $B$ that lie outside the blue/light-shaded areas are set to $\infty$. We set $A^{(\ell)}[i] := (1+\varepsilon)^d$ and $B^{(\ell)}[j] := B[j] + (1+\varepsilon)^d$. This guarantees the approximation $(1-\varepsilon)(A[i]+B[j]) \leq \max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \leq (1+\varepsilon)(A[i]+B[j])$ for close pairs. Numbers in non-overlapping parts differ by so much that their sum and their max are equal up to a factor $1+\varepsilon$. This ensures that they do not interfere with the close pairs.

**Lemma 6.3.2** (Close Covering)**.** *Given vectors $A, B \in \mathbb{R}_+^n$ and a parameter $\varepsilon > 0$, there are vectors $A^{(1)}, \ldots, A^{(s)}, B^{(1)}, \ldots, B^{(s)} \in \mathbb{R}_+^n$ with $s = \mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ such that:*

(i) *for all* $i, j \in [n]$ *and all* $\ell \in [s]$:

$$\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \geq (1-\varepsilon)(A[i] + B[j]), \text{ and}$$

(ii) *for all* $i, j \in [n]$ *if* $\frac{A[i]}{B[j]} \in [\varepsilon, 1/\varepsilon]$ *then* $\exists \ell \in [s]$:

$$\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \leq (1+\varepsilon)(A[i] + B[j]).$$

*We can compute such vectors* $A^{(1)}, \ldots, A^{(s)}, B^{(1)}, \ldots, B^{(s)}$ *in time* $\mathcal{O}(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$.

*Proof.* We choose $s = \Theta(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ with sufficiently large hidden constant, and for any $\ell \in \{1, \ldots, s\}$ construct vectors $A^{(\ell)}, B^{(\ell)}$ as described in Algorithm 9.

**Algorithm 9** CloseCovering$(A, B, \varepsilon)$.

1: Set $s := 1 + \lceil 2 \log_{1+\varepsilon}(1/\varepsilon) \rceil$ ▷ Note that $s = \Theta(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$.
2: **for** $\ell = 1, \ldots, s$ **do** ▷ Take care of $A[i] \approx (1+\varepsilon)^{ks+\ell}$ for any $k$.
3: $D_\ell := \{ks + \ell \mid k \in \mathbb{Z}\}$
4: $A^{(\ell)}[i] := \begin{cases} (1+\varepsilon)^d & \text{if } A[i] \in \left[(1+\varepsilon)^{d-1}, (1+\varepsilon)^d\right) \text{ for } d \in D_\ell \\ \infty & \text{otherwise} \end{cases}$
5: $B^{(\ell)}[j] := \begin{cases} B[j] + (1+\varepsilon)^d & \text{if } B[j] \in \left[\varepsilon(1+\varepsilon)^{d-1}, \frac{1}{\varepsilon}(1+\varepsilon)^d\right) \text{ for } d \in D_\ell \\ \infty & \text{otherwise} \end{cases}$
6: **end for**
7: **return** $\{(A^{(1)}, B^{(1)}), \ldots, (A^{(s)}, B^{(s)})\}$

Note that the condition for $B[j]$ is well-defined in the sense that it applies for at most one $d \in D_\ell$. To see this, since two consecutive values in $D_\ell$ differ by $s$, we only need to show the inequality $\frac{1}{\varepsilon}(1+\varepsilon)^d \leq \varepsilon(1+\varepsilon)^{d+s-1}$, which holds since $s \geq 1 + \log_{1+\varepsilon}(1/\varepsilon^2)$. The same can be immediately seen to hold for $A[i]$.

The size and time bounds are immediate. It remains to prove correctness.

For property (ii), consider any $i, j$ with $\frac{A[i]}{B[j]} \in [\varepsilon, 1/\varepsilon]$. Note that there is a unique $\ell \in \{1, \ldots, s\}$ such that $A^{(\ell)}[i] \neq \infty$. For this $\ell$, we have $A^{(\ell)}[i] = (1+\varepsilon)^d$ with $(1+\varepsilon)^{d-1} \leq A[i] < (1+\varepsilon)^d$, for some $d \in D_\ell$. By the assumption $\frac{A[i]}{B[j]} \in [\varepsilon, 1/\varepsilon]$, we obtain $\varepsilon(1+\varepsilon)^{d-1} \leq B[j] \leq \frac{1}{\varepsilon}(1+\varepsilon)^d$, and thus $B^{(\ell)}[j]$ is not set to $\infty$, and we have $B^{(\ell)}[j] = B[j] + (1+\varepsilon)^d$. We conclude by observing that $\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} = B^{(\ell)}[j] = B[j] + (1+\varepsilon)^d \leq (1+\varepsilon)(B[j] + A[j])$.

For property (i), consider any $i, j$ and $\ell$. If one of $A^{(\ell)}[i], B^{(\ell)}[j]$ is set to $\infty$, then the property holds trivially. Otherwise, we have $A^{(\ell)}[i] = (1+\varepsilon)^d$ for some $d \in D_\ell$ and $B^{(\ell)}[j] = B[j] + (1+\varepsilon)^{d'}$ for some $d' \in D_\ell$. We consider two cases.

*Case 1:* $d \leq d'$. Then $A[i] \leq (1+\varepsilon)^d \leq (1+\varepsilon)^{d'}$, and thus $B^{(\ell)}[j] = B[j] + (1+\varepsilon)^{d'} \geq A[i] + B[j]$.

*Case 2:* $d > d'$. Then by definition of $D_\ell$ we have $d \geq d' + s$. We bound

$$B[j] \leq \tfrac{1}{\varepsilon}(1+\varepsilon)^{d'} \leq \tfrac{1}{\varepsilon}(1+\varepsilon)^{d-s} \leq \tfrac{1}{\varepsilon}(1+\varepsilon)^{1-s}A[i] \leq \varepsilon A[i], \tag{6.1}$$

where the last inequality uses $s \geq 1 + \log_{1+\varepsilon}(1/\varepsilon^2)$. This yields

$$A^{(\ell)}[i] \geq A[i] \overset{(6.1)}{\geq} (1-\varepsilon)A[i] + B[j] \geq (1-\varepsilon)(A[i] + B[j]).$$

In both cases we have $\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \geq (1-\varepsilon)(A[i] + B[j])$, which proves property (i). □

### 6.3.2 Distant Covering

We now want to cover all pairs $i, j$ with $\frac{A[i]}{B[j]} \notin [\varepsilon, 1/\varepsilon]$. Our solution for this case is similar to the well-known *Well-Separated Pair Decomposition* (see [13, 33]), which we use in a one-dimensional setting and in log-scale. The main difference is that we unite sufficiently distant pairs of the decomposition that lie on the same level.

Our constructed vectors $A^{(\ell)}$ will correspond to subsets of the entries of $A$, i.e., we have $A^{(\ell)}[i] \in \{A[i], \infty\}$, and similarly for $B$. For this reason, we switch to subset notation for the majority of this section, and then return to our usual notation of vectors $A^{(\ell)}, B^{(\ell)}$ in Corollary 6.3.10.

For $x, y \in \mathbb{R}_+$, we define their *distance* as $d(x,y) := \max\{\frac{x}{y}, \frac{y}{x}\}$ if $x, y < \infty$ and $d(x,\infty) = d(\infty, x) = \infty$ otherwise. For sets $X, Y \subset \mathbb{R}_+$, we define their *distance* as $d(X,Y) := \min_{x\in X, y\in Y} d(x,y)$.

**Lemma 6.3.3** (Distant Covering, Set Variant)**.** *Given a set $Z \subset \mathbb{R}_+$ of size $n$ and a parameter $\varepsilon > 0$, there are sets $X_1, \ldots, X_s \subseteq Z$ and $Y_1, \ldots, Y_s \subseteq Z$ with $s = \mathcal{O}(\log n \log \frac{1}{\varepsilon})$ such that:*

(i) *for any $\ell \in [s]$ we have $d(X_\ell, Y_\ell) > \frac{1}{\varepsilon}$, and*

(ii) *for any $x, y \in Z$ with $d(x,y) \geq \frac{2}{\varepsilon}$ and $x < y$ there is $\ell \in [s]$ such that $x \in X_\ell$ and $y \in Y_\ell$.*

*We can compute sets $X_1, \ldots, X_s$ and $Y_1, \ldots, Y_s$ satisfying (1) and (2) time $\mathcal{O}(n \log n \log \frac{1}{\varepsilon})$.*

We will later use $Z$ as the set of all entries of vectors $A$ and $B$. Regarding (i), observe that if $d(x,y) > \frac{1}{\varepsilon}$, then the sum $x + y$ and the maximum $\max\{x, y\}$ differ by less than a factor $1 + \varepsilon$. This allows us to ensure point

(i) of Sum-to-Max-Covering. Property (ii) ensures that we cover all distant pairs and thus corresponds to point (ii) of Sum-to-Max-Covering.

The proof outline is as follows, see also Algorithm 10 for pseudocode. To simplify notation we assume $n$ to be a power of 2 (this is without loss of generality since we can fill up $Z$ with arbitrary numbers). We first sort $Z$, so from now on we assume that $Z = \{z_1, \ldots, z_n\}$ with $z_1 \leq \ldots \leq z_n$. The algorithm performs $\log n$ iterations. In iteration $r$, we split $Z$ into chunks of size $n/2^r$, and we remove some chunks that are irrelevant for covering distant pairs, see procedure SplitChunks and Figure 6.2. Then we separate the resulting list of chunks into two sub-lists, see procedure SeparateChunks and Figure 6.3. Finally, we handle the transition between any two chunks by introducing a restricted area at their boundary, applied with $\mathcal{O}(\log \frac{1}{\varepsilon})$ different shifts, see procedure ShiftedTransitions and Figure 6.4. In the following subsections we describe the individual procedures in detail.

**Algorithm 10** DistantCoveringLemma$(Z, \varepsilon)$

```
sort(Z)                                          ▷ Z = {z_1 ≤ z_2 ≤ ... ≤ z_n}
Set T_0 as a list containing one element, T_0[1] := Z
for r = 1, 2, ..., ⌈log n⌉ do
    T_r := SplitChunks(T_{r-1}, ε)
    T_{r,1}, T_{r,2} := SeparateChunks(T_r)
    S_{r,1} := ShiftedTransitions(T_{r,1}, ε)
    S_{r,2} := ShiftedTransitions(T_{r,2}, ε)
end for
return ⋃_r S_{r,1} ∪ S_{r,2}
```

### SplitChunks

Algorithm 11 describes the procedure of selecting chunks $T_r$ in every level, see also Figure 6.2 for an illustration. We start with a big chunk $T_0[1] = Z$, containing the whole input. Then we iterate over all levels $r = 1, 2, \ldots, \log n$ and construct refined chunks as follows. In iteration $r$, we iterate over all previous chunks $T_{r-1}[i]$. If $T_{r-1}[i]$ does not contain any two numbers in distance greater than $\frac{1}{\varepsilon}$, then we can ignore it. Otherwise, we split $T_{r-1}[i]$ at the middle into two chunks of half the size and add them to the list of chunks $T_r$. For any $r$, this yields a list of chunks $T_r$ such that

(P1) every chunk $T_r[i]$ is a subset of $Z$ of the form $\{z_a, z_{a+1}, \ldots, z_b\}$ and of size $|T_r[i]| = n/2^r$, and

(P2) every $x \in T_r[i]$ is smaller than every $y \in T_r[j]$, for any $i < j$.

Note that at the bottom level, chunks have size 1. Moreover, for any $r > 0$ the list $T_r$ contains an even number of chunks; this will also hold for all lists of chunks constructed later.

**Algorithm 11** SplitChunks$(T_{r-1}[1 \dots \ell], \varepsilon)$

```
Initialize T_r as an empty list, and k := 1
for i = 1, 2, ..., ℓ do
    By construction, T_{r-1}[i] is of the form {z_a, z_{a+1}, ..., z_b} for some a ≤ b
    if z_a < ε · z_b then
        T_r[2k − 1] := {z_a, ..., z_{(a+b−1)/2}}
                                          ▷ Split T_{r-1}[i] in the middle
        T_r[2k] := {z_{(a+b+1)/2}, ..., z_b}
        k := k + 1
    end if
end for
return T_r
```

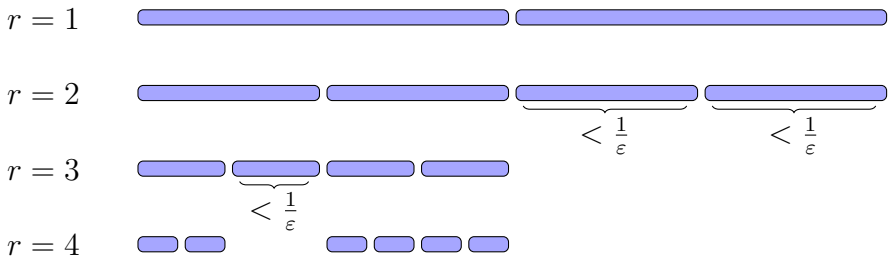



Figure 6.2: Illustration of the procedure SplitChunks, which splits and selects chunks of the input numbers on different levels $r$.

**Claim 6.3.4.** *We have* $d\big(T_r[2k], T_r[2k+3]\big) > \frac{1}{\varepsilon}$ *for any level* $r$ *and any index* $k$.

*Proof.* Write the parent chunk $T_r[2k+1] \cup T_r[2k+2]$ in the form $\{z_a, z_{a+1}, \dots, z_b\}$. Since $T_r[2k+1], T_r[2k+2]$ have been added, we have $z_a < \varepsilon \cdot z_b$. Since every number in $T[2k]$ is smaller than $z_a$ and every number in $T[2k+3]$ is larger than $z_b$, the distance of $T[2k], T[2k+3]$ is greater than $\frac{1}{\varepsilon}$. □

The main property of our splitting procedure is that all $x, y \in Z$ with $d(x, y) > \frac{1}{\varepsilon}$ eventually are contained in consecutive chunks (see Figure 6.2) – note that we will only make use of consecutive chunks with indices $2k-1, 2k$ for some $k$ (as opposed to $2k, 2k+1$).

**Claim 6.3.5.** *For any $x, y \in Z$, if $d(x,y) > \frac{1}{\varepsilon}$ and $x < y$, then there exist a level $r$ and index $k$ such that $x \in T_r[2k-1]$ and $y \in T_r[2k]$.*

*Proof.* Consider the largest $r$ such that $x, y$ are contained in the same chunk $T_r[k']$. The chunk $T_r[k']$ contains at least two elements, so $r < \log n$. Since $d(x,y) > \frac{1}{\varepsilon}$, Algorithm 11 splits $T_r[k']$ into chunks $T_{r+1}[2k-1]$ and $T_{r+1}[2k]$. By maximality of $r$ and by $x < y$, it follows that $x \in T_{r+1}[2k-1]$ and $y \in T_{r+1}[2k]$. This proves the claim. □

### SeparateChunks

The procedure SeparateChunks is given a list $T_r$ of chunks and separates it into two subsequences $T_{r,1}$ and $T_{r,2}$, where $T_{r,1}$ contains all chunks $T[i]$ with $(i \bmod 4) \in \{1,2\}$, and $T_{r,2}$ contains the remaining chunks in $T$. See Algorithm 12 for pseudocode and Figure 6.3 for an illustration.

**Algorithm 12** SeparateChunks($T_r[1 \dots 2\ell]$)

```
1: Initialize T_{r,1}, T_{r,2} as empty lists, and b := 1
2: for k = 1, 2, ..., ℓ do
3:     Append T_r[2k − 1] and T_r[2k] to T_{r,b}
4:     b := 3 − b
5: end for
6: return T_{r,1}, T_{r,2}
```

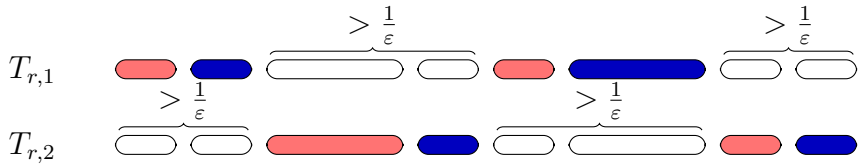



Figure 6.3: Illustration of SeparateChunks, which separates the list of chunks $T_r$ on some level $r$ into two sub-lists $T_{r,1}$ and $T_{r,2}$. The selected chunks are marked in red/light-shaded and blue/dark-shaded. (In the next step, the red/light-shaded chunks will form a set $X_\ell$, and the blue/dark-shaded ones will form a set $Y_\ell$. Note that within $T_{r,1}$ every red chunk is $\varepsilon$-distant from every blue chunk, except for its right neighbor. This will be used by the procedure ShiftedTransitions.)

This construction ensures that consecutive chunks with indices $2k$ and $2k+1$ have distance at least $\frac{1}{\varepsilon}$, as shown in the following claim.

**Claim 6.3.6.** *We have* $d\Big(T_{r,b}[2k], T_{r,b}[2k+1]\Big) > \frac{1}{\varepsilon}$ *for any level* $r$*, index* $k$*, and* $b \in \{1,2\}$*.*

*Proof.* Because of how $T_{r,1}, T_{r,2}$ are constructed, the chunks $T_{r,b}[2k]$ and $T_{r,b}[2k+1]$ correspond to chunks $T_r[2k']$ and $T_r[2k'+3]$ for some $k'$. The statement now follows from Claim 6.3.4. □

The following analogue of Claim 6.3.5 is immediate.

**Claim 6.3.7.** *For any* $x, y \in Z$*, if* $d(x,y) > \frac{1}{\varepsilon}$ *and* $x < y$*, then there exist a level* $r$*, index* $k$*, and* $b \in \{1,2\}$ *such that* $x \in T_{r,b}[2k-1]$ *and* $y \in T_{r,b}[2k]$*.*

*Proof.* Consecutive chunks $T_r[2k-1]$ and $T_r[2k]$ are either both added to $T_{r,1}$ or both added to $T_{r,2}$. The statement thus follows from Claim 6.3.5. □

### ShiftedTransitions

The procedure ShiftedTransitions is given a list of chunks $T = T_{r,b}$ and returns $\mathcal{O}(\log \frac{1}{\varepsilon})$ many pairs $(X_t, Y_t)$ of the final covering (recall the statement of Lemma 6.3.3). Naively, we would like to assign every odd chunk to $X_t$ and every even chunk to $Y_t$, i.e., $X_t = \bigcup_k T[2k-1]$ and $Y_t = \bigcup_k T[2k]$. From Claim 6.3.6 we know that even chunks are distant from their right neighbors, i.e., $d(T[2k], T[2k+1]) > \frac{1}{\varepsilon}$. This is not necessarily true for $d(T[2k-1], T[2k])$, and therefore we introduce a restricted area at their boundary, applied with $\mathcal{O}(\log \frac{1}{\varepsilon})$ different shifts, as illustrated in Figure 6.4. See Algorithm 13.

**Algorithm 13** ShiftedTransitions$(T[1,\dots 2\ell], \varepsilon)$

```
for t = 0, 1, ..., ⌈log_2 1/ε⌉ do
    for k ∈ {1, ..., ℓ} do
        let z_min be the minimal number in T[2k]
        T'[2k−1] := {z ∈ T[2k−1] | z ≤ ε 2^t z_min}
        T'[2k] := {z ∈ T[2k] | z > 2^t z_min}
    end for
    X_t := ⋃_k T'[2k−1]
    Y_t := ⋃_k T'[2k]
end for
return {(X_t, Y_t) | 0 ≤ t ≤ ⌈log_2 1/ε⌉}
```

**Claim 6.3.8.** *For any output* $(X_t, Y_t)$ *by* ShiftedTransitions$(T_{r,b}, \varepsilon)$ *we have* $d(X_t, Y_t) > \frac{1}{\varepsilon}$*.*

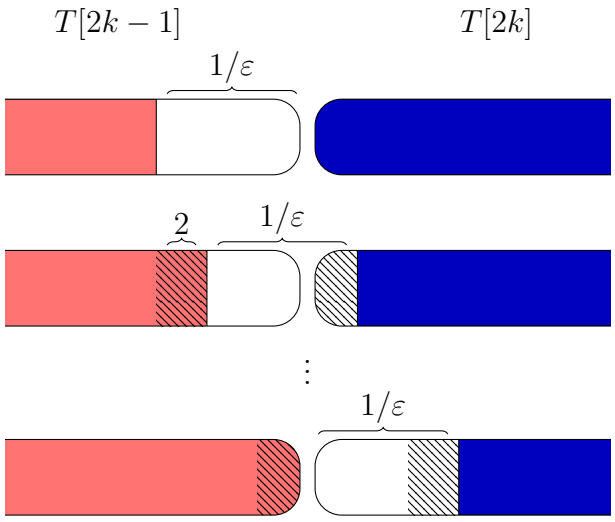



Figure 6.4: Illustration of ShiftedTransitions in log-scale. Dashed areas represent the added/removed numbers from iteration $t$ to $t+1$. In each iteration, we shift to the right by a factor 2, resulting in at most $\left\lceil \log_2 \frac{1}{\varepsilon} \right\rceil$ iterations. Note that in each iteration the red/light-shaded numbers are in distance greater than $\frac{1}{\varepsilon}$ from the blue/dark-shaded numbers. Moreover, the distance between any two numbers in the dashed area is less than $\frac{2}{\varepsilon}$.

*Proof.* Let $T = T_{r,b}$. By Claim 6.3.6 and sortedness (see property (P2)), any chunks $T[i]$ and $T[j]$ with $j \geq i+2$ have distance greater than $\frac{1}{\varepsilon}$. Since $X_t$ only contains numbers from odd chunks $T[2k-1]$, and $Y_t$ only contains numbers from even chunks $T[2k]$, we obtain that any $x \in X_t, y \in Y_t$ within distance $\frac{1}{\varepsilon}$ satisfy $x \in T[2k-1], y \in T[2k]$ for some index $k$. However, in any iteration $t$ the subsets $T'[2k-1] \subseteq T[2k-1]$ and $T'[2k] \subseteq T[2k]$ are chosen to have distance greater than $\frac{1}{\varepsilon}$, and hence $d(X_t, Y_t) > \frac{1}{\varepsilon}$. □

**Claim 6.3.9.** *For any $x, y \in Z$, if $d(x,y) \geq \frac{2}{\varepsilon}$ and $x < y$, then there exist a level $r$ and $b \in \{1,2\}$ such that* ShiftedTransitions$(T_{r,b}, \varepsilon)$ *outputs a pair $(X_t, Y_t)$ with $x \in X_t, y \in Y_t$.*

*Proof.* By Claim 6.3.7, there are $r, k, b$ with $x \in T_{r,b}[2k-1]$ and $y \in T_{r,b}[2k]$. Let $T = T_{r,b}$ and let $z_{\min}$ be the minimal number in $T[2k]$. If $x \leq \varepsilon \cdot z_{\min}$, then in iteration $t = 0$ we construct $T'[2k-1]$ containing $x$, and $T'[2k] = T[2k]$ contains $y$, so $x \in X_0$ and $y \in Y_0$. Otherwise, let $t \in \mathbb{N}$ be minimal with $x \leq \varepsilon 2^t \cdot z_{\min}$. By sortedness (see property (P2)) we have $x < z_{\min}$ and thus

$t \leq \left\lceil \log_2 \frac{1}{\varepsilon} \right\rceil$. Hence, in iteration $t$ the set $T'[2k-1]$ contains $x$. Moreover, by minimality of $t$ and $d(x,y) \geq \frac{2}{\varepsilon}$ we have $\varepsilon 2^{t-1} \cdot z_{\min} < x \leq \frac{\varepsilon}{2} y$, and thus $y > 2^t \cdot z_{\min}$, so $y$ is contained in $T'[2k]$, yielding $x \in X_t, y \in Y_t$. □

### Proof of Distant Covering

*Proof of Lemma 6.3.3.* Properties (i) and (ii) follow immediately from Claims 6.3.8 and 6.3.9. The bound $s = \mathcal{O}(\log n \log \frac{1}{\varepsilon})$ on the number of constructed pairs $(X_\ell, Y_\ell)$ is immediate from the loops in Algorithms 10 and 13. Finally, the running time of $\mathcal{O}(n \log n \log \frac{1}{\varepsilon})$ is immediate from inspecting the pseudocode. □

It remains to translate Lemma 6.3.3 to the vector notation of Sum-to-Max-Covering.

**Corollary 6.3.10** (Distant Covering, Vector Variant)**.** *Given vectors $A, B \in \mathbb{R}_+^n$ and a parameter $\varepsilon > 0$, there are vectors $A^{(1)}, \ldots, A^{(s)}$, $B^{(1)}, \ldots, B^{(s)} \in \mathbb{R}_+^n$ with $s = \mathcal{O}(\log n \log \frac{1}{\varepsilon})$ such that:*

(i) *for all $i, j \in [n]$ and all $\ell \in [s]$:*

$$\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \geq (1-2\varepsilon)(A[i] + B[j]), \quad \textit{and}$$

(ii) *for all $i, j \in [n]$ if $\frac{A[i]}{B[j]} \notin [\varepsilon, 1/\varepsilon]$ then $\exists \ell \in [s]$:*

$$\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \leq A[i] + B[j].$$

*We can compute such vectors $A^{(1)}, \ldots, A^{(s)}, B^{(1)}, \ldots, B^{(s)}$ in time $\mathcal{O}(n \log n \log \frac{1}{\varepsilon})$.*

*Proof.* Set $Z := \{A[i], B[i] \mid i \in [n]\}$ and run Lemma 6.3.3 on $(Z, \varepsilon')$ with $\varepsilon' := 2\varepsilon$ to obtain subsets $X_1, \ldots, X_s \subseteq Z$ and $Y_1, \ldots, Y_s \subseteq Z$ with $s = \mathcal{O}(\log n \log \frac{1}{\varepsilon})$. We only double the number of subsets by considering $(X'_1, \ldots, X'_{2s}) := (X_1, \ldots, X_s, Y_1, \ldots, Y_s)$ and $(Y'_1, \ldots, Y'_{2s}) := (Y_1, \ldots, Y_s, X_1, \ldots, X_s)$. We construct vectors $A^{(\ell)}, B^{(\ell)}$ with $\ell \in [2s]$ by setting

$$A^{(\ell)}[i] := \begin{cases} A[i] & \text{if } A[i] \in X'_\ell, \\ \infty & \text{otherwise,} \end{cases} \qquad B^{(\ell)}[i] := \begin{cases} B[i] & \text{if } B[i] \in Y'_\ell, \\ \infty & \text{otherwise.} \end{cases}$$

The size and time bounds are immediate.

For any numbers $x, y \in \mathbb{R}_+$ with $d(x,y) > \frac{1}{\varepsilon}$ we claim that $\max\{x,y\} \geq (1-\varepsilon)(x+y)$. Indeed, assume without loss of generality $x < \varepsilon y$, then we have $\max\{x,y\} = y > (1-\varepsilon)y + x \geq (1-\varepsilon)(x+y)$.

Property (i) of Lemma 6.3.3 yields that $d(A^{(\ell)}[i], B^{(\ell)}[j]) > \frac{1}{\varepsilon'}$ for any $i, j, \ell$. Hence, we have

$$\begin{aligned}\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} &\geq (1-\varepsilon')(A^{(\ell)}[i] + B^{(\ell)}[j]) \\ &\geq (1-\varepsilon')(A[i] + B[j]) = (1-2\varepsilon)(A[i] + B[j]),\end{aligned}$$

proving property (i) of the corollary.

Note that by switching from $X_1, \ldots, X_s$ and $Y_1, \ldots, Y_s$ to $X'_1, \ldots, X'_{2s}$ and $Y'_1, \ldots, Y'_{2s}$ we made Lemma 6.3.3 symmetric, and thus the requirement $x < y$ of its property (ii) is removed. Thus, property (ii) of Lemma 6.3.3 yields that for any $i, j$ with $d(A[i], B[j]) \geq \frac{2}{\varepsilon'} = \frac{1}{\varepsilon}$ there exists $\ell \in [2s]$ with $A^{(\ell)}[i] = A[i]$ and $B^{(\ell)}[j] = B[j]$, and thus

$$\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} = \max\{A[i], B[j]\} \leq A[i] + B[j],$$

proving property (ii) of the corollary. □

### 6.3.3 Proof of Sum-To-Max-Covering

The following variant of Sum-To-Max-Covering follows immediately from combining Close Covering (Lemma 6.3.2) and Distant Covering (Lemma 6.3.3). Note that compared to Lemma 6.3.1 there is an additional factor $1 - 2\varepsilon$ on the right hand side of property (i).

**Lemma 6.3.11** (Weaker Sum-to-Max-Covering)**.** *Given vectors $A, B \in \mathbb{R}^n_+$ and a parameter $\varepsilon > 0$, there are vectors $A^{(1)}, \ldots, A^{(s)}, B^{(1)}, \ldots, B^{(s)} \in \mathbb{R}^n_+$ with $s = \mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon} + \log n \log\frac{1}{\varepsilon})$ such that:*

(i) *for all $i, j \in [n]$ and all $\ell \in [s]$:*

$$\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \geq (1-2\varepsilon)(A[i] + B[j]), \quad \textit{and}$$

(ii) *for all $i, j \in [n]$ there exists $\ell \in [s]$:*

$$\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \leq (1+\varepsilon)(A[i] + B[j]).$$

*We can compute such vectors $A^{(1)}, \ldots, A^{(s)}, B^{(1)}, \ldots, B^{(s)}$ in time $\mathcal{O}(\frac{n}{\varepsilon}\log\frac{1}{\varepsilon} + n \log n \log\frac{1}{\varepsilon})$.*

*Proof of Lemma 6.3.11.* Given vectors $A, B \in \mathbb{R}^n_+$ and a parameter $\varepsilon > 0$, we simply run Close Covering and Distant Covering on $(A, B, \varepsilon)$ and concatenate the results, see Algorithm 14. Correctness as well as size and time bounds are immediate consequences of Lemma 6.3.2 and Corollary 6.3.10. □

**Algorithm 14** WeakerSumToMaxCovering$(A, B, \varepsilon)$.

1: **return** DistantCovering$(A, B, \varepsilon) \cup$ CloseCovering$(A, B, \varepsilon)$

*Proof of Lemma 6.3.1.* To prove the stronger variant given in the beginning of Section 6.3, we have to remove the factor $1 - 2\varepsilon$ on the right hand side of property (i) in Lemma 6.3.11. To this end, given $A, B, \varepsilon$, we run the construction from Lemma 6.3.11 on $A, B, \varepsilon'$ with $\varepsilon' := \frac{\varepsilon}{5}$, and we divide every entry of the resulting vectors by $1 - 2\varepsilon'$, see Algorithm 15. For correctness, note that the division by $1-2\varepsilon'$ removes the factor $1-2\varepsilon'$ from property (i) in Lemma 6.3.11, i.e., we obtain the claimed $\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \geq A[i] + B[j]$ for all $i, j, \ell$. For property (ii), note that the division by $1-2\varepsilon'$ leaves us with $\max\{A^{(\ell)}[i], B^{(\ell)}[j]\} \leq \frac{1+\varepsilon'}{1-2\varepsilon'}(A[i] + B[j])$ for all $i, j$ and some $\ell$. Using $\varepsilon' = \frac{\varepsilon}{5}$ and $\frac{1+\varepsilon/5}{1-2\varepsilon/5} \leq 1 + \varepsilon$ for any $\varepsilon \in (0, 1]$ finishes the proof. □

**Algorithm 15** SumToMaxCovering$(A, B, \varepsilon)$.

1: **return** $\frac{1}{1-2\varepsilon/5} \cdot$ WeakerSumToMaxCovering$(A, B, \frac{\varepsilon}{5})$

## 6.4 Strongly polynomial approximation for undirected APSP

In this section, we present a strongly polynomial $(1 + \varepsilon)$-approximation for APSP on undirected graphs that runs in time $\widetilde{\mathcal{O}}(n^\omega/\varepsilon)$. Our algorithm will use the scaling technique, but we combine it with edge contractions and amortized analysis to avoid the factor $\log W$; this is inspired by Tardos [144] and uses similar edge contraction arguments as [44, 101]. We start by describing the previously fastest approximation algorithm for APSP (Section 6.4.1), and then present our adaptations (Section 6.4.2).

### 6.4.1 Zwick's approximation for APSP

Zwick [170] obtained an $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon} \log W)$-time $(1 + \varepsilon)$-approximation algorithm for (directed or undirected) APSP as follows. The first step is to reduce approximate APSP to approximate Min-Plus Product, see Theorem 6.1.1. It is well-known that Min-Plus Product on $n \times n$-matrices with entries in $\{1, \ldots, W\}$ can be solved exactly in time $\widetilde{\mathcal{O}}(Wn^\omega)$. Zwick utilized this fact via *adaptive scaling* to realize his second step, as shown in Algorithm 17.

**Algorithm 16** SCALE$(A, q, \varepsilon)$.

1: $A'[i,j] = \begin{cases} \lceil A[i,j]/(\varepsilon 2^q) \rceil & \text{if } A[i,j] \leq 2^q \\ \infty & \text{otherwise} \end{cases}$
2: **return** $A'$

**Algorithm 17** ZWICK-APX-MINPROD$(A, B, \varepsilon)$.

1: Initialize $\tilde{C}[i,j] := \infty$ for all $i, j$
2: Let $W$ be the largest entry of $A, B$
3: **for** $q = 0, 1, 2, \ldots, \lceil \log W \rceil + 1$ **do**
4: $A' = \text{SCALE}(A, q, \varepsilon)$ ▷ Scale matrix $A$ so entries are from $\{0, \ldots, \left\lceil \frac{1}{\varepsilon} \right\rceil\}$
5: $B' = \text{SCALE}(B, q, \varepsilon)$
6: $C' = \text{MINPLUSPROD}(A', B')$ ▷ This works in time $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon})$
7: $\tilde{C}[i,j] = \min\{C[i,j], \varepsilon 2^q C'[i,j]\}$ for all $i, j$
8: **end for**
9: **return** $\tilde{C}$

In each iteration $q$ of Algorithm 17 the entries that are greater than $2^q$ are ignored (they are replaced by $\infty$). The remaining entries are scaled and rounded to lie in the range $\{1, \ldots, \left\lceil \frac{1}{\varepsilon} \right\rceil\}$ by Algorithm 16. This yields scaled matrices $A', B'$ with integer entries bounded by $\mathcal{O}(\frac{1}{\varepsilon})$, so their Min-Plus Product $C'$ can be computed in time $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon})$. The output $\tilde{C}$ of the algorithm is the entry-wise minimum of all computed products $C'$ over all $q$. The total running time is $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon} \log W)$.

Since we always round up, one can see that each entry of $\tilde{C}$ is at least the corresponding entry of the correct Min-Plus Product $C$ of $A, B$. For the other direction, for all $i, j$ there is an iteration $q$ such that $2^{q-1} < C[i,j] \leq 2^q$. Consider any $k$ with $C[i,j] = A[i,k] + B[k,j]$. Then $A[i,k], B[k,j] \leq 2^q$, so they are not set to $\infty$. We obtain that $C'[i,j] \leq A'[i,k] + B'[k,j] \leq \frac{A[i,k]+B[k,j]}{\varepsilon 2^q} + 2$, and thus $\tilde{C}[i,j] \leq \varepsilon 2^q C'[i,j] \leq C[i,j] + \varepsilon 2^{q+1} \leq (1+4\varepsilon) C[i,j]$. Replacing $\varepsilon$ by $\varepsilon/4$ yields the claimed approximation.

### 6.4.2 Undirected APSP in strongly polynomial matrix-multiplication time

We now essentially remove the $\log W$-factor from Zwick's algorithm for undirected graphs, proving the following restated theorem from the introduction.

**Theorem 1.4.2.** *$(1+\varepsilon)$-Approximate Undirected APSP is in strongly polynomial time*[1] $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon})$.

*Proof of Theorem 1.4.2.* The algorithm proceeds in iterations $q = 1, 2, 4, \ldots$ up to the largest power of 2 bounded by $nW$. In iteration $q$, the goal is to find all shortest paths of length in $[q, 2q)$. For this, all edges of $G$ of weight at least $2q$ are irrelevant. Therefore, in the first iteration we start with an empty graph $H$, and in iteration $q$ we add all edges of $G$ with weight in $[q, 2q)$ to $H$. We then round down all edge weights to multiples of $q\varepsilon/n$. This may result in edges of weight 0, which we contract (this crucially uses that we consider undirected graphs). Finally, we run Zwick's algorithm on $H$ and update the corresponding distances. Specifically, if we compute a distance $\tilde{d}_H(u,v) \in [(1-\varepsilon)q, (1+\varepsilon)2q)$ in $H$, then we iterate over all vertices $i$ in $G$ that were contracted to $u$ in $H$, and similarly over all $j$ that were contracted to $v$, and we update our estimated distance $D[i,j]$. See Algorithm 18.

**Algorithm 18** ApproximateUndirectedAPSP$(G, \varepsilon)$.

1: Initialize $H$ to be the graph with $n$ isolated nodes, i.e., $H = (V(G), \emptyset)$
2: Initialize $D[i,j] := \infty$ for all $i, j \in V(G)$
3: **for** $q = 1, 2, 4, \ldots, 2^{\lfloor \log(nW) \rfloor}$ **do** ▷ Find all shortest paths of length in $[q, 2q)$:
4: Add all edges of $G$ with weight in $[q, 2q)$ to $H$
5: Round down all edge weights of $H$ to multiples of $\frac{q\varepsilon}{n}$
6: Contract all edges of $H$ with weight 0
7: Run Zwick's $(1+\varepsilon)$-approximation for APSP on $H$, obtaining distances $\tilde{d}_H(u,v)$
8: **for** all nodes $u, v$ of $H$ with $\tilde{d}_H(u,v) \in [(1-\varepsilon)q, (1+\varepsilon)2q)$ **do**
9: $D[i,j] := \min\{D[i,j], \tilde{d}_H(u,v)\}$ for every node $i$ ($j$) of $G$ that was contracted to $u$ ($v$),
10: **end for**
11: **end for**
12: Return $D$

**Correctness** Denote by $d_G(i,j)$ the correct distance between $i$ and $j$ in $G$, and by $d_H(.,.)$ the correct distance in $H$. The computed approximation satisfies $d_H(u,v) \leq \tilde{d}_H(u,v) \leq (1+\varepsilon)d_H(u,v)$ for any $u, v \in V(H)$.

[1]Here, by time we mean the number of *arithmetic operations* performed on a RAM machine. The *bit complexity* of the algorithm is bounded by the number of arithmetic operations times $\log\log W$ (up to terms hidden by $\widetilde{\mathcal{O}}$).

**Claim 6.4.1.** *Consider $i, j \in V(G)$ that have been contracted to $u, v$ in $H$, respectively. If we have $\tilde{d}_H(u,v) \geq (1-\varepsilon)q$ then $\tilde{d}_H(u,v) \geq (1-\varepsilon)d_G(i,j)$.*

Since we only update distances when $\tilde{d}_H(u,v) \in [(1-\varepsilon)q, (1+\varepsilon)2q)$, by this claim the output of Algorithm 18 satisfies $D[i,j] \geq (1-\varepsilon)d_G(i,j)$ for all $i, j \in V(G)$.

*Proof of Claim 6.4.1.* Consider a path $P$ from $u$ to $v$ in $H$ realizing $d_H(u,v)$. Uncontract all contracted edges of $H$ to obtain a graph $H'$. Since we only contracted edges of (rounded) weight 0, the path $P$ corresponds to a path $P'$ from $i$ to $j$ in $H'$ of (rounded) length $d_H(u,v)$. Since we can assume $P'$ to be a simple path and we rounded down edge weights to multiples of $q\varepsilon/n$, in the graph $G$ path $P'$ has length at most $d_H(u,v) + q\varepsilon$. Hence, we have $d_G(i,j) \leq d_H(u,v) + q\varepsilon$.

Note that the claim is trivial if $d_G(i,j) \leq q$, so assume $d_G(i,j) > q$. Then we conclude by bounding $\tilde{d}_H(u,v) \geq d_H(u,v) \geq d_G(i,j) - q\varepsilon > (1-\varepsilon)d_G(i,j)$. □

It remains to show $D[i,j] \leq (1+\varepsilon)d_G(i,j)$ for all $i, j \in V(G)$. Consider the iteration $q$ with $d_G(i,j) \in [q, 2q)$. Note that all edges of any shortest path between $i$ and $j$ are added in or before iteration $q$. Let $u, v$ be the nodes that $i, j$ have been contracted to in $H$ in iteration $q$, respectively. Since rounding down to multiples of $q\varepsilon/n$ reduces the distance by at most $q\varepsilon$, and since contracting edges of rounded weight 0 does not change any distances, we have $d_H(u,v) \in [(1-\varepsilon)d_G(i,j), d_G(i,j)]$. This yields $d_H(u,v) \in [(1-\varepsilon)q, 2q)$, and in particular we have $u \neq v$. By the properties of the approximation (i.e., $d_H(u,v) \leq \tilde{d}_H(u,v) \leq (1+\varepsilon)d_H(u,v)$), we obtain $\tilde{d}_H(u,v) \in [(1-\varepsilon)q, (1+\varepsilon)2q)$. This triggers the update of $D[i,j]$, which yields $D[i,j] \leq \tilde{d}_H(u,v) \leq (1+\varepsilon)d_H(u,v) \leq (1+\varepsilon)d_G(i,j)$.

In total, we obtain that $(1-\varepsilon)d_G(i,j) \leq D[i,j] \leq (1+\varepsilon)d_G(i,j)$ for all $i, j \in V(G)$. By dividing all output entries $D[i,j]$ by $(1-\varepsilon)$, we may instead obtain a "one-sided" approximation of the form $d_G(i,j) \leq D'[i,j] \leq (1+\mathcal{O}(\varepsilon))d_G(i,j)$.

**Running Time** We call an iteration $q$ *void* if $G$ has no edge with weight in $[q, 2q)$ and $H$ is empty (i.e., $H$ consists of isolated vertices). Observe that void iterations do not change $H$ or $D$. In order to avoid the $\log W$-factor of a naive implementation of Algorithm 18, we skip over all void iterations.

In iteration $q$, let $C_{q,1}, \ldots, C_{q,k(q)}$ be all connected components of $H$ of size at least 2, and let $n_{q,1}, \ldots, n_{q,k(q)}$ be their sizes (i.e., number of vertices). Note that instead of running Zwick's algorithm on $H$, it suffices to run it on each $C_{q,i}$. Moreover, after rounding, the ratio of the largest to smallest

weight in $H$ is $\mathcal{O}(n/\varepsilon)$, and thus the $\log W$ factor of Zwick's algorithm is $\mathcal{O}(\log(n/\varepsilon))$. Hence, we can bound the contribution of Zwick's algorithm to our running time by $\widetilde{\mathcal{O}}\Big(\sum_{q,i}(n_{q,i}^{\omega}/\varepsilon)\log(n/\varepsilon)\Big)$.

The life-cycle of every pair of vertices $u, v \in V(G)$ can be described by the following states: (1) $u$ and $v$ are not connected in $H$; (2) $u$ and $v$ are connected in $H$; (3) $u$ and $v$ are contracted into the same vertex of $H$. Observe that each pair $u, v \in V(G)$ can be in state (2) for at most $\mathcal{O}(\log(n/\varepsilon))$ iterations. Indeed, if $u$ and $v$ are connected in iteration $q$, then in iteration $q' > \frac{n}{\varepsilon}q$ they have been contracted. It follows that $\sum_{q,i} n_{q,i}^2 = \mathcal{O}(n^2 \log(n/\varepsilon))$, since the former counts the pairs of connected nodes in $H$, summed over all iterations $q$.

Combining this with our running time bound of $\widetilde{\mathcal{O}}\Big(\sum_{q,i}(n_{q,i}^{\omega}/\varepsilon)\log(n/\varepsilon)\Big)$, the fact $\omega \geq 2$, and Jensen's inequality, yields a time bound of $\widetilde{\mathcal{O}}(n^{\omega}/\varepsilon)$. This bounds the running time spent in calls to Zwick's algorithm. Most other parts of our algorithm can be seen to take time $\widetilde{\mathcal{O}}(n^2)$ in total.

A subtle point is the update of matrix entries $D[i, j]$ in line 9 of Algorithm 18. Consider any $i, j$ and let $u, v$ be the vertices that $i, j$ have been contracted to in $H$ in iteration $q$, respectively. Note that $D[i, j]$ can only change if $u \neq v$ and $u$ and $v$ are connected in $H$. This situation can only happen for $\mathcal{O}(\log(n/\varepsilon))$ iterations, since then $u$ and $v$ will be contracted to the same vertex. Hence, in total updating $D$ takes time $\mathcal{O}(n^2 \log(n/\varepsilon))$. This finishes the proof. □

## 6.5 Strongly polynomial approximation for graph characteristics

One of the fundamental challenges in network science is the identification of "important" or "central" nodes in a network. Different *graph characteristics* have been proposed to capture this notion [64]. For example the *Median* of a graph is a node that minimizes the sum of the distances to all other nodes in graph, the *Center* of a graph is a node that minimizes the maximum distance to any other node (this distance is called Radius) and the *Diameter* of a graph is the distance of the furthest pair of vertices in the graph. Centrality measures are actively generalized to weighted graphs [121]. In this section, we present a simple argument that yields strongly polynomial approximation schemes for these problems. The following theorem is restated from the introduction.

**Theorem 1.4.1.** *Diameter, Radius, Median, Minimum-Weight Triangle,*

*and Minimum-Weight Cycle on directed and undirected graphs have approximation schemes in strongly polynomial time* $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon})$.

Abboud, Grandoni, and Williams [4] observed that *Diameter, Radius* and *Median* admit $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon}\log W)$-time approximation schemes via Zwick's approximation of APSP. Similarly, Roditty and Williams [128] observed that *Minimum Weight Triangle* admits an $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon}\log W)$-time approximation scheme. They used this as a black-box to show that *Minimum Weight Cycle* (both in directed and undirected graphs) admits an $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon}\log W)$-time approximation scheme.

*Proof of Theorem 1.4.1.* Let $G$ be a given graph. For any number $w \in \mathbb{R}_+$, define $G_w$ as the graph $G$ where we remove all edges of weight $> w$ and change the weight of all remaining edges to $w$. On $G_w$ we can compute a 2-approximation for each of the considered problems in time $\widetilde{\mathcal{O}}(n^\omega)$ (since $\varepsilon = 1$ is constant and there are only two different edge weights, so also W is constant). Note that if the result on $G_w$ is infinite, then the solution value on $G$ is greater than $w$, as we need to include at least one edge of weight greater than $w$. Moreover, if the solution value on $G_w$ is finite, then it is at most $w \cdot n^2$, since this is the total weight of all edges in $G_w$. In particular, this means that the solution value of $G$ is at most $wn^2$.

We use this as follows. First, we sort all edge weights of $G$ and perform binary search to determine the smallest edge weight $w$ of $G$ such that the solution value on $G_w$ is finite. It follows that the solution value on $G$ is in $[w, wn^2]$, so we have an $\mathcal{O}(n^2)$-approximation. Now we round up all edge weights of $G$ to multiples of $w\varepsilon/n^2$. This changes the total edge weight of $G$ by at most $\varepsilon w$, and thus also the weight of an optimal solution by at most a multiplicative factor $1+\varepsilon$. The ratio between the largest and smallest weight in the resulting graph $G'$ is at most $W \leq \frac{n^4}{\varepsilon}$. Hence the $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon}\log W)$-time approximation scheme runs in time $\widetilde{\mathcal{O}}(\frac{n^\omega}{\varepsilon})$ on $G'$. □

## 6.6 Strongly polynomial approximation for Min-Plus Convolution

In this section, we consider sequences $A \in \mathbb{R}_+^n$ and index their entries by $A[0], \ldots, A[n-1]$. Recall, that given two sequences $A, B \in \mathbb{R}_+^n$, the *convolution problem in the* $(\otimes, \oplus)$*-semiring* is to compute the sequence $C \in \mathbb{R}_+^n$ with $C[k] = \bigoplus_{i+j=k}(A[i] \otimes B[j])$ for all $0 \leq k < n$. Clearly, the problem can be solved using $\mathcal{O}(n^2)$ ring operations. In the standard $(\cdot, +)$-ring, the problem can be solved in time $\mathcal{O}(n \log n)$ by Fast Fourier Transform (FFT).

In this section, we start with a simple $(1+\varepsilon)$-approximation algorithm for Min-Plus Convolution that directly follows from our Sum-To-Max-Covering and runs in time $\widetilde{\mathcal{O}}(n^{3/2}/\varepsilon)$, see Theorem 6.6.1. This is the first strongly polynomial $(1+\varepsilon)$-approximation algorithm for this problem (with a running time of $\mathcal{O}(n^{2-\delta})$ for any $\delta > 0$). We then prove an equivalence of approximate Min-Plus Convolution and exact Min-Max Convolution, see Theorem 6.6.2. Finally, we use more problem-specific arguments to obtain an improved approximation algorithm running in time $\widetilde{\mathcal{O}}(n^{3/2}/\varepsilon^{1/2})$, see Theorem 6.6.3.

### 6.6.1 Simple approximation algorithm

Direct application of our Sum-to-Max-Covering yields the following strongly polynomial approximation algorithm for Min-Plus Convolution, similarly as for APSP.

**Theorem 6.6.1.** *$(1+\varepsilon)$-Approximate Min-Plus Convolution can be solved in time $\widetilde{\mathcal{O}}(n^{3/2}/\varepsilon)$.*

*Proof.* Consider input sequences $(A[i])_{i=0}^{n-1}$, $(B[i])_{i=0}^{n-1}$ on which we want to compute $(C[k])_{k=0}^{n-1}$ with $C[k] = \min_{i+j=k}(A[i] + B[j])$. We view the sequences $A, B$ as vectors in $\mathbb{R}_+^n$, in order to apply Sum-To-Max-Covering (Lemma 6.3.1). This yields sequences $A^{(1)}, \ldots, A^{(s)}$, $B^{(1)}, \ldots, B^{(s)} \in \mathbb{R}_+^n$ with $s = \mathcal{O}(\varepsilon^{-1} \operatorname{polylog}(n/\varepsilon))$. We compute the Min-Max Convolution of every layer $A^{(\ell)}, B^{(\ell)}$. (Note that we can first replace the entries of $A^{(\ell)}, B^{(\ell)}$ by their ranks; this is necessary since the input format for approximate Min-Plus Convolution is floating-point, but Min-Max Convolution requires standard bit representation of integers.) We then return the entry-wise minimum of the results, see Algorithm 19. The output sequence $\tilde{C}$ is a $(1+\varepsilon)$-approximation of $C$; this follows from the properties of Sum-To-Max-Covering, analogously to the proof of Theorem 6.1.2. Since Sum-To-Max-Covering takes time $\widetilde{\mathcal{O}}(n/\varepsilon)$, the running time is dominated by computing $s$ times a Min-Max Convolution, resulting in $\widetilde{\mathcal{O}}(sn^{3/2}) = \widetilde{\mathcal{O}}(n^{3/2}/\varepsilon)$. □

**Algorithm 19** ApproximateMinConv$(A, B, \varepsilon)$

1: $\{(A^{(1)}, B^{(1)}), \ldots, (A^{(s)}, B^{(s)})\} = \text{SumToMaxCovering}(A, B, \varepsilon)$
2: $C^{(\ell)} := \text{MinMaxConv}(A^{(\ell)}, B^{(\ell)})$ for any $\ell \in [s]$
3: $\tilde{C}[k] := \min_{\ell \in [s]}\{C^{(\ell)}[k]\}$ for any $0 \le k < n$
4: **return** $\tilde{C}$

### 6.6.2 Equivalence of approximate Min-Plus and Exact Min-Max Convolution

We next show an equivalence of approximate Min-Plus Convolution and exact Min-Max Convolution, similarly to the equivalence for matrix products in Theorem 1.4.4.

**Theorem 6.6.2.** *For any $c \geq 1$, if one of the following statements is true, then all are:*

- *$(1+\varepsilon)$-Approximate Min-Plus Convolution can be solved in strongly polynomial time $\widetilde{\mathcal{O}}(n^c/\mathrm{poly}(\varepsilon))$,*
- *exact Min-Max Product can be solved in strongly polynomial time $\widetilde{\mathcal{O}}(n^c)$,*

In particular, any further improvement on the exponent of $n$ in Theorem 6.6.1 would yield an improved algorithm for Min-Max Convolution.

*Proof.* For one direction, observe that if Min-Max Convolution can be solved in time $T(n)$ then the algorithm from Theorem 6.6.1 runs in time $\widetilde{\mathcal{O}}(n/\varepsilon + T(n)/\varepsilon)$, which is $\widetilde{\mathcal{O}}(T(n))$ for constant $\varepsilon > 0$.

For the other direction, on input $A, B$ denote the result of Min-Max Convolution by $C$. Set $r := \lceil 4(1+\varepsilon)^2 \rceil$ and consider the sequences $A', B'$ with $A'[i] := r^{A[i]}$ and $B'[j] := r^{B[j]}$. (Note that the integers $A[i], B[j]$ are in standard bit representation, so we can compute floating-point representations of $r^{A[i]}, r^{B[j]}$ in constant time, essentially by writing $A[i], B[j]$ into the exponent.) Let $C'$ be the result of $(1+\varepsilon)$-Approximate Min-Plus Convolution on $A', B'$. Then as in the proof of Theorem 1.4.4, we obtain $r^{C[k]} \leq C'[k] \leq r^{C[k]+1/2}$. Hence, we can infer the Min-Max Convolution $C$ of $A, B$ by setting $C[k] = \lfloor \log_r C'[k] \rfloor$ (i.e., we essentially only read the exponent of the floating-point number $C'[k]$). If $(1+\varepsilon)$-Approximate Min-Plus Convolution is in time $T(n)$, this yields an algorithm for Min-Max Convolution running in time $\widetilde{\mathcal{O}}(n + T(n)) = \widetilde{\mathcal{O}}(T(n))$. □

### 6.6.3 Improved approximation algorithm

In the remainder of this section, we improve the simple approximation algorithm of Theorem 6.6.1.

**Theorem 6.6.3.** *$(1+\varepsilon)$-Approximate Min-Plus Convolution can be solved in time $\widetilde{\mathcal{O}}(n^{3/2}/\sqrt{\varepsilon})$.*

We will divide the algorithm for Min-Plus Convolution into two parts: the first part will handle the case when summands are *close* and the second will handle the *distant* case.

### Approximating Min-Plus Convolution for distant summands

First, we will simply use Distant Covering (Corollary 6.3.10) to correctly compute Min-Plus Convolution for summands that differ by at least a factor $\frac{1}{\varepsilon}$.

**Lemma 6.6.4.** *Given sequences $A, B \in \mathbb{R}^n_+$ and a parameter $\varepsilon > 0$, let $C$ be the result of Min-Plus Convolution on $A, B$. In $\mathcal{O}(n^{3/2} \operatorname{polylog}(\frac{n}{\varepsilon}))$ time we can compute a sequence $\tilde{C}$ such that:*

(i) *for any $k \in [n]$ we have $\tilde{C}[k] \geq C[k]$, and*

(ii) *if there are $i + j = k$ with $C[k] = A[i] + B[j]$ and $\frac{A[i]}{B[j]} \notin [\frac{\varepsilon}{4}, \frac{4}{\varepsilon}]$ then $\tilde{C}[k] \leq (1 + \varepsilon)C[k]$.*

**Algorithm 20** DistantMinConv$(A, B, \varepsilon)$

1: $\{(A^{(1)}, B^{(1)}), \ldots, (A^{(s)}, B^{(s)})\} =$ DistantCovering$(A, B, \varepsilon/4)$
2: $C^{(\ell)} :=$ MinMaxConv$(A^{(\ell)}, B^{(\ell)})$ for any $\ell \in [s]$
3: $\tilde{C}[k] := \frac{1}{1-\varepsilon/2} \cdot \min_{\ell \in [s]}\{C^{(\ell)}[k]\}$ for any $0 \leq k < n$
4: **return** $\tilde{C}$

*Proof of Lemma 6.6.4.* We view the sequences $A, B$ as vectors in $\mathbb{R}^n_+$, and apply Distant Covering (Corollary 6.3.10) with $\varepsilon' := \frac{\varepsilon}{4}$. This yields sequences $A^{(1)}, \ldots, A^{(s)}$, $B^{(1)}, \ldots, B^{(s)} \in \mathbb{R}^n_+$ with $s = \mathcal{O}(\operatorname{polylog}(\frac{n}{\varepsilon}))$. We compute the Min-Max Convolution of every layer $A^{(\ell)}, B^{(\ell)}$. Then we compute the entry-wise minimum of the results, scale it by a factor $\frac{1}{1-2\varepsilon'}$, and return the resulting sequence $\tilde{C}$, see Algorithm 20.

For correctness, note that the scaling factor $\frac{1}{1-2\varepsilon'}$ removes the factor $1-2\varepsilon'$ from the right hand side of property (i) in Corollary 6.3.10. This yields $\tilde{C}[k] \geq C[k]$. Moreover, by property (ii) of Corollary 6.3.10, for any indices $i+j = k$ with $\frac{A[i]}{B[j]} \notin [\varepsilon', \frac{1}{\varepsilon'}] = [\frac{\varepsilon}{4}, \frac{4}{\varepsilon}]$ there is an $\ell$ such that $C^{(\ell)}[k] \leq A[i]+B[j]$. Minimizing over all $\ell$ and multiplying by $\frac{1}{1-2\varepsilon'} = \frac{1}{1-\varepsilon/2} < 1 + \varepsilon$ yields the claimed property (ii).

Since Distant Covering takes time $\mathcal{O}(n \operatorname{polylog}(\frac{n}{\varepsilon}))$, the running time is dominated by computing $s$ times Min-Max Convolution. Using the fastest known algorithm for Min-Max Convolution [103], we obtain time $\widetilde{\mathcal{O}}(n^{3/2}s) = \mathcal{O}(n^{3/2} \operatorname{polylog}(\frac{n}{\varepsilon}))$. □

### Approximating Min-Plus Convolution for close summands

We now use a variant of a known scaling-based approximation scheme for Min-Plus Convolution to handle the close summands.

**Lemma 6.6.5.** *Given sequences $A, B \in \mathbb{R}^n_+$ and a parameter $\varepsilon > 0$, let $C$ be the result of Min-Plus Convolution on $A, B$. In $\tilde{\mathcal{O}}(n^{3/2}/\sqrt{\varepsilon})$ time we can compute a sequence $\tilde{C}$ such that:*

(i) *for any $k \in [n]$ we have $\tilde{C}[k] \geq C[k]$, and*

(ii) *if there are $i + j = k$ with $C[k] = A[i] + B[j]$ and $\frac{A[i]}{B[j]} \in [\frac{\varepsilon}{4}, \frac{4}{\varepsilon}]$ then $\tilde{C}[k] \leq (1 + \varepsilon)C[k]$.*

**Algorithm 21** SCALE$(A, q, \varepsilon)$.

$A'[i] = \begin{cases} \left\lceil \frac{4}{\varepsilon q} \cdot A[i] \right\rceil & \text{if } \frac{\varepsilon q}{16} \leq A[i] \leq q \\ \infty & \text{otherwise} \end{cases}$

**return** $A'$

**Algorithm 22** CLOSEMINCONV$(A, B, \varepsilon)$.

Initialize $\tilde{C}[k] := \infty$ for all $k$
**for** $q = 1, 2, 4, \ldots, 2^{\lceil \log 2W \rceil}$ **do**
    $A' :=$ SCALE$(A, q, \varepsilon)$
    $B' :=$ SCALE$(B, q, \varepsilon)$
    $V :=$ EXACTMINCONV$(A', B')$
    $\tilde{C}[k] := \min\{\tilde{C}[k], V[k] \cdot \frac{q\varepsilon}{4}\}$ for all $k$
**end for**
**return** $\tilde{C}$

*Proof.* Consider the procedures SCALE and CLOSEMINCONV (Algorithms 21 and 22)We claim that this algorithm proves Lemma 6.6.5. Correctness is based on the rounding lemma (we restate it here).

**Lemma 5.7.3** (Rounding Lemma)**.** *For natural numbers $x, y$ and positive $q, \varepsilon$ satisfying $q \leq x + y$ and $0 < \varepsilon < 1$ it holds:*

$$\begin{aligned} x + y &\leq \left( \left\lceil \frac{2x}{q\epsilon} \right\rceil + \left\lceil \frac{2y}{q\epsilon} \right\rceil \right) \frac{q\epsilon}{2} &< (x + y)(1 + \epsilon), \\ (x + y)(1 - \epsilon) &< \left( \left\lfloor \frac{2x}{q\epsilon} \right\rfloor + \left\lfloor \frac{2y}{q\epsilon} \right\rfloor \right) \frac{q\epsilon}{2} &\leq x + y. \end{aligned}$$

**Correctness** Correctness of CloseMinConv can now be shown similarly as in Section 5.7. Regarding property (i), the lower bound of Lemma 5.7.3 ensures that we have $\tilde{C}[k] \geq C[k]$ throughout the run of the algorithm. Regarding property (ii), for any $i+j=k$ with $C[k] = A[i]+B[j]$ there exists a precision parameter $q$ with $q/2 \leq A[i] + B[j] \leq q$. In particular, we have $A[i], B[j] \leq q$ and $\max\{A[i], B[j]\} \geq \frac{q}{4}$. If we additionally have $\frac{A[i]}{B[j]} \in [\frac{\varepsilon}{4}, \frac{4}{\varepsilon}]$, then

$$\min\{A[i], B[j]\} \geq \tfrac{\varepsilon}{4} \cdot \max\{A[i], B[j]\} \geq \tfrac{\varepsilon q}{16},$$

and thus $A'[i]$ and $B'[j]$ both are not set to $\infty$ by the procedure Scale. The upper bound of Lemma 5.7.3 now implies $\tilde{C}[k] \leq (1+\varepsilon)(A[i]+B[j]) = (1+\varepsilon)C[k]$.

**Running time** For the running time analysis, denote by $\alpha_q$ the number of entries of $A'$ that are not set to $\infty$ in iteration $q$. Note that if an entry $A[i]$ is not set to $\infty$ in iteration $q$, then we have $\frac{\varepsilon q}{16} \leq A[i] \leq q$, or, equivalently, $A[i] \leq q \leq \frac{16}{\varepsilon} A[i]$. Since $q$ grows geometrically, there are $\mathcal{O}(\log \frac{1}{\varepsilon})$ iterations $q$ in which entry $A[i]$ is not set to $\infty$. Hence, we obtain $\sum_q \alpha_q = \mathcal{O}(n \log \frac{1}{\varepsilon})$. We similarly define $\beta_q$ as the number of non-$\infty$ entries of $B'$ in iteration $q$, and obtain

$$\sum_q \beta_q = \mathcal{O}\Big(n \log \tfrac{1}{\varepsilon}\Big). \tag{6.2}$$

We argue in the following that procedure CloseMinConv can be implemented in such a way that the running time for iteration $q$ is

$$\mathcal{O}(\min\{\alpha_q\beta_q, \frac{n}{\varepsilon}\} \operatorname{polylog}(\frac{n}{\varepsilon})).$$

To this end, we use an event queue to be able to skip all iterations with $\alpha_q = 0$ or $\beta_q = 0$. Moreover, we can maintain all non-$\infty$ entries of $A', B'$ in time $\mathcal{O}(\alpha_q + \beta_q)$ per iteration $q$. Note that $\mathcal{O}(\alpha_q + \beta_q) \leq \mathcal{O}(\min\{\alpha_q\beta_q, \frac{n}{\varepsilon}\})$. This yields the claimed time bound for lines 3 and 4 of procedure CloseMinConv.

For line 5 of procedure CloseMinConv, note that naively the exact Min-Plus Convolution of $A'$ and $B'$ can be computed in time $\mathcal{O}(\alpha_q\beta_q)$. Moreover, since $A', B'$ have entries in $\{1, \ldots, W\} \cup \{\infty\}$ for $W = \left\lceil \frac{4}{\varepsilon} \right\rceil$, by Fast Fourier Transform their Min-Plus Convolution can be computed in time $\widetilde{\mathcal{O}}(Wn) = \widetilde{\mathcal{O}}(\frac{n}{\varepsilon})$. Using the better of the two yields the claimed time bound.

Finally, line 6 of procedure CloseMinConv can be implemented in time $\mathcal{O}(\min\{\alpha_q\beta_q, n\})$, since this is an upper bound on the number of non-$\infty$ entries of $V$.

Altogether, we obtain time $\mathcal{O}(\min\{\alpha_q\beta_q, \frac{n}{\varepsilon}\}\operatorname{polylog}(\frac{n}{\varepsilon}))$ per iteration $q$. Hence, the total running time of procedure CloseMinConv is bounded by

$$\sum_q \min\left\{\alpha_q\beta_q, \tfrac{n}{\varepsilon}\right\}\operatorname{polylog}(\tfrac{n}{\varepsilon}).$$

We split this sum into the cases $\beta_q \leq \lambda$ and $\beta_q > \lambda$, and note that the second case can occur at most $\mathcal{O}(\frac{n}{\lambda}\log\frac{1}{\varepsilon})$ times due to (6.2). This yields a total running time of at most

$$\left(\left(\sum_q \alpha_q\lambda\right) + \tfrac{n}{\varepsilon}\cdot\tfrac{n}{\lambda}\log\tfrac{1}{\varepsilon}\right)\operatorname{polylog}(\tfrac{n}{\varepsilon}) \leq \left(n\lambda + \tfrac{n^2}{\varepsilon\lambda}\right)\operatorname{polylog}(\tfrac{n}{\varepsilon}).$$

Setting $\lambda := (n/\varepsilon)^{1/2}$ yields the claimed total running time bound $\widetilde{\mathcal{O}}(n^{3/2}/\varepsilon^{1/2})$. □

**Proof of Theorem 6.6.3**

**Algorithm 23** ApxMinConv$(A, B, \varepsilon)$.

1: $\tilde{C}_1 :=$ DistantMinConv$(A, B, \varepsilon)$
2: $\tilde{C}_2 :=$ CloseMinConv$(A, B, \varepsilon)$
3: $\tilde{C}[k] := \min\{\tilde{C}_1[k], \tilde{C}_2[k]\}$ for any $0 \leq k < n$
4: **return** $\tilde{C}$

*Proof of Theorem 6.6.3.* Given sequences $A, B \in \mathbb{R}^n_+$ and a parameter $\varepsilon > 0$, we simply run our algorithms for approximating Min-Plus Convolution on distant and close summands, and compute their entry-wise minimum (see Algorithm 23). Correctness as well as size and time bounds are immediate consequences of Lemmas 6.6.4 and 6.6.5. □

### 6.6.4 Applications for Tree Sparsity

A direct consequence of the strongly polynomial $(1+\varepsilon)$-approximation for Min-Plus Convolution is an analogous strongly polynomial $(1+\varepsilon)$-approximation for Tree Sparsity. We will make use of the strongly polynomial reduction that uses an approximation algorithm for Min-Plus Convolution as a black-box that we have showed in Chapter 5.

**Theorem 5.8.1.** *If $(1+\varepsilon)$-approximate Min-Plus Convolution can be solved in time $T(n, W, \varepsilon)$, then $(1+\varepsilon)$-approximate Tree Sparsity can be solved in time $\mathcal{O}\Big(\Big(n + T(n, W, \varepsilon/\log^2 n)\Big)\log n\Big)$.*

Note, that this reduction runs in strongly polynomial time, and the $\log W$-factors in the running time of [18] come exclusively from the use of scaling for approximating Min-Plus Convolution. In consequence, our strongly polynomial $(1+\varepsilon)$-approximation for Min-Plus Convolution yields a strongly polynomial $(1+\varepsilon)$-approximation for Tree Sparsity.

**Corollary 6.6.6.** $(1+\varepsilon)$*-Approximate Tree Sparsity can be solved in time* $\widetilde{\mathcal{O}}(\frac{n^{3/2}}{\sqrt{\varepsilon}})$.

Tree Sparsity has applications in image processing, computational biology [138] and machine learning [18]. The main issue of previous approximation schemes for this problem was that in applications the input consists of real numbers and thus $\log W$-factors significantly influence the running time [132].

# Part IV

# Conclusion

# Chapter 7

# Conclusion and future work

In this chapter, we summarize the thesis and propose future research directions.

## 7.1 Exact graph algorithms

In Chapter 3, we introduced the framework based on Frobenius normal form and used it to solve some problems on directed, unweighted graphs in matrix multiplication time. The main open question is to use this framework to prove that APSP on such graphs can be solved in $\widetilde{\mathcal{O}}(n^\omega)$ or at least $\mathcal{O}(n^{2.5})$. The promising way is to use the algorithms that determine operators of matrices of polynomials (e.g., determinant, solving linear system [78, 118]). Additionally, algorithms for a black-box polynomial degree determination seem to be a promising path.

Another interesting problem is to use this framework to obtain additive approximation for APSP. Currently, the best additive approximation of APSP is due to [127]. However, no additive approximation of APSP is known that would work in $\widetilde{\mathcal{O}}(n^\omega)$ time.

Application in dynamic algorithm also seems to be a promising approach. Frandsen and Sankowski [61] showed an algorithm, that dynamically preserves Frobenius normal form in $\mathcal{O}(kn^2)$ time. Our algorithms use fast Hankel matrix-vector multiplication that is based on FFT. Reif and Tate [125] presented an $\mathcal{O}(\sqrt{n})$ time per request algorithm for FFT. Can we use these approaches to obtain a faster dynamic algorithm?

Finally, it remains open how to apply the Frobenius normal form in the weighted directed graphs with small, integer weights $\{-W, \ldots, W\}$. Cygan, Gabow, and Sankowski [49] took degree $W$ polynomials and used Mulders and Storjohann [118] algorithms to get $\widetilde{\mathcal{O}}(Wn^\omega)$ time radius and diameter

detection. We suspect that similar technique can be applied to Frobenius normal form framework.

## 7.2 Connecting dynamic programming with approximation

In Chapter 4, we mainly study the complexity of the Partition problem (which is related to Knapsack and Subset Sum). In the exact setting, if we are only concerned about the dependence on $n$, Knapsack and Subset Sum were already known to be equivalent up to the polynomial factors. Nederlof, Leeuwen, and Zwaan [120, Theorem 2] showed, that if there exists an exact algorithm for Subset Sum working in $\mathcal{O}^*(T(n))$ time and $\mathcal{O}^*(S(n))$ space, then we can construct an algorithm for Knapsack working in the same $\mathcal{O}^*(T(n))$ time and $\mathcal{O}^*(S(n))$ space. In contrast, in the realm of pseudo-polynomial time complexity, Subset Sum seems to be simpler than Knapsack (see Bringmann [25]). We show similar separation for Knapsack and Partition in the approximation setting.

The most important question is whether one can significantly improve the running time of our algorithm for Partition to $\widetilde{\mathcal{O}}(n + 1/\varepsilon)$. One can also ask whether randomization is necessary to obtain subquadratic FPTAS for Partition. Perhaps, the randomized building blocks can be replaced with deterministic algorithms by Kellerer, Pferschy, and Speranza [100] and Koiliaris and Xu [102].

## 7.3 Equivalences in tropical convolutions

In Chapter 5, we undertake a systematic study of Min-Plus Convolution as a hardness assumption and prove the subquadratic equivalence of Min-Plus Convolution with SuperAdditivity Testing, Unbounded Knapsack, Knapsack, and Tree Sparsity. The Min-Plus Convolution conjecture is stronger than the well-known conjectures APSP and 3SUM. Proving that Min-Plus Convolution is equivalent to either APSP or 3SUM would solve a long-standing open problem. An intriguing question is to determine whether the Min-Plus Convolution conjecture is also stronger than OV.

By using the fast $\mathcal{O}(n^2/2^{\Omega(\log n)^{1/2}})$ algorithm for Max-Plus Convolution, we automatically obtain $o(n^2)$-time algorithms for all problems in the class. This gives us the first (to the best of our knowledge) subquadratic algorithm for SuperAdditivity Testing and improves exact algorithms for Tree Sparsity.

One consequence of our results is a new lower bound on Knapsack. It is known that an $\mathcal{O}(t^{1-\epsilon}n^{\mathcal{O}(1)})$ algorithm for Knapsack contradicts the SetCover conjecture [47]. Here, we show that an $\mathcal{O}((n+t)^{2-\epsilon})$ algorithm contradicts the Min-Plus Convolution conjecture. This does not rule out an $\mathcal{O}(t+n^{\mathcal{O}(1)})$ algorithm, which leads to another interesting open problem.

Recently, Abboud et al. [3] replaced the SetCover conjecture with the SETH for Subset Sum. We have shown that one cannot exploit the SETH to prove that the $\mathcal{O}(nt)$-time algorithm for Unbounded Knapsack is tight. The analogous question regarding Knapsack remains open.

Finally, it is open whether Max-Plus Convolution LowerBound is equivalent to Min-Plus Convolution, which would imply an equivalence between $l_\infty$-Necklace Alignment and Min-Plus Convolution.

## 7.4 Equivalence of approximate tropical product

In Chapter 6, we prove the equivalence of computing the approximate Min-Plus Product of two $n \times n$ matrices with arbitrary values and Min-Max Product. This results in a subcubic strongly polynomial approximation for APSP. Our algorithm runs in time $\mathcal{O}(n^{\frac{\omega+3}{2}}\varepsilon^{-1}\text{polylog}(n,\varepsilon)) \leq \widetilde{\mathcal{O}}(\frac{n^{2.69}}{\varepsilon})$ on word RAM. We also give efficient approximation algorithms for undirected APSP, computing graph characteristic (e.g., radius, diameter, minimum weight cycle, etc.) and show that our equivalence results also apply to convolutions.

The main open problem is to improve upon $\widetilde{\mathcal{O}}(n^{\frac{\omega+3}{2}})$ algorithm for APBP or give some evidence that $\mathcal{O}(n^{2.5-\delta})$ is unlikely (for any constant $\delta > 0$). Another interesting open problem is to improve upon $\mathcal{O}(n^{1.5}\sqrt{\log n})$ algorithm for Min-Max Convolution or prove that $\mathcal{O}(n^{1.5-\delta})$ is unlikely (for any constant $\delta > 0$).

# Bibliography


[1] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. "Tight Hardness Results for LCS and Other Sequence Similarity Measures". In: *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. Ed. by Venkatesan Guruswami. IEEE Computer Society, 2015, pp. 59–78.

[2] Amir Abboud and Greg Bodwin. "The 4/3 Additive Spanner Exponent Is Tight". In: *J. ACM* 64.4 (2017), 28:1–28:20.

[3] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. "SETH-Based Lower Bounds for Subset Sum and Bicriteria Path". In: *arXiv preprint arXiv:1704.04546, to appear at SODA 2019* (2019).

[4] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. "Subcubic Equivalences Between Graph Centrality Problems, APSP and Diameter". In: *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*. 2015, pp. 1681–1697.

[5] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. "Subcubic Equivalences Between Graph Centrality Problems, APSP and Diameter". In: *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*. Ed. by Piotr Indyk. SIAM, 2015, pp. 1681–1697.

[6] Amir Abboud and Aviad Rubinstein. "Fast and Deterministic Constant Factor Approximation Algorithms for LCS Imply New Circuit Lower Bounds". In: *Proc. 9th Innovations in Theoretical Computer Science Conference (ITCS'18)*. Vol. 94. 2018, 35:1–35:14.

[7] Amir Abboud, Aviad Rubinstein, and R. Ryan Williams. "Distributed PCP Theorems for Hardness of Approximation in P". In: *Proc. 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS'17)*. 2017, pp. 25–36.

[8] Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. "Approximation and Fixed Parameter Subquadratic Algorithms for Radius and Diameter in Sparse Graphs". In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*. Ed. by Robert Krauthgamer. SIAM, 2016, pp. 377–391.

[9] Udit Agarwal and Vijaya Ramachandran. "Fine-grained complexity for sparse graphs". In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*. Ed. by Ilias Diakonikolas, David Kempe, and Monika Henzinger. ACM, 2018, pp. 239–252.

[10] Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. "On the Complexity of Numerical Analysis". In: *SIAM J. Comput.* 38.5 (2009), pp. 1987–2006.

[11] Noga Alon, Raphael Yuster, and Uri Zwick. "Color-Coding". In: *J. ACM* 42.4 (1995), pp. 844–856.

[12] Noga Alon, Raphael Yuster, and Uri Zwick. "Finding and Counting Given Length Cycles". In: *Algorithmica* 17.3 (1997), pp. 209–223.

[13] Sunil Arya, Gautam Das, David M. Mount, Jeffrey S. Salowe, and Michiel H. M. Smid. "Euclidean spanners: short, thin, and lanky". In: *Proc. 27th Annual ACM Symposium on Theory of Computing (STOC'95)*. 1995, pp. 489–498.

[14] Per Austrin, Petteri Kaski, Mikko Koivisto, and Jussi Määttä. "Space-Time Tradeoffs for Subset Sum: An Improved Worst Case Algorithm". In: *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*. Ed. by Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg. Vol. 7965. Lecture Notes in Computer Science. Springer, 2013, pp. 45–56.

[15] Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. "Subset Sum in the Absence of Concentration". In: *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*. Ed. by Ernst W. Mayr and Nicolas Ollinger. Vol. 30. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 48–61.

[16] Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. "Dense Subset Sum May Be the Hardest". In: *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*. Ed. by Nicolas Ollinger and Heribert Vollmer. Vol. 47. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 13:1–13:14.

[17] Arturs Backurs and Piotr Indyk. "Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)". In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*. Ed. by Rocco A. Servedio and Ronitt Rubinfeld. ACM, 2015, pp. 51–58.

[18] Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. "Better Approximations for Tree Sparsity in Nearly-Linear Time". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2017, pp. 2215–2229.

[19] Nikhil Bansal, Shashwat Garg, Jesper Nederlof, and Nikhil Vyas. "Faster space-efficient algorithms for subset sum and k-sum". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. Ed. by Hamed Hatami, Pierre McKenzie, and Valerie King. ACM, 2017, pp. 198–209.

[20] Ilya Baran, Erik D. Demaine, and Mihai Patrascu. "Subquadratic Algorithms for 3SUM". In: *Algorithmica* 50.4 (2008), pp. 584–596.

[21] Richard Bellman. *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1957.

[22] Anand Bhalgat, Ashish Goel, and Sanjeev Khanna. "Improved Approximation Results for Stochastic Knapsack Problems". In: *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*. Ed. by Dana Randall. SIAM, 2011, pp. 1647–1665.

[23] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Springer Science & Business Media, 2012.

[24] David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, and Perouz Taslakian. "Necklaces, Convolutions, and X + Y". In: *Algorithms – ESA 2006: 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006. Proceedings*. Ed. by Yossi Azar and Thomas

Erlebach. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 160–171.

[25] Karl Bringmann. "A near-linear pseudopolynomial time algorithm for subset sum". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2017, pp. 1073–1084.

[26] Karl Bringmann. personal communication. April 2018.

[27] Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. "Truly Sub-cubic Algorithms for Language Edit Distance and RNA-Folding via Fast Bounded-Difference Min-Plus Product". In: *Proc. IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS'16)*. 2016, pp. 375–384.

[28] Karl Bringmann and Marvin Künnemann. "Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping". In: *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. Ed. by Venkatesan Guruswami. IEEE Computer Society, 2015, pp. 79–97.

[29] Karl Bringmann, Marvin Künnemann, and Karol Wegrzycki. "Approximating APSP without scaling: equivalence of approximate min-plus and exact min-max". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*. Ed. by Moses Charikar and Edith Cohen. ACM, 2019, pp. 943–954.

[30] Peter Burcsi, Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. "On Table Arrangements, Scrabble Freaks, and Jumbled Pattern Matching". In: *Fun with Algorithms, 5th International Conference, FUN 2010, Ischia, Italy, June 2-4, 2010. Proceedings*. Ed. by Paolo Boldi and Luisa Gargano. Vol. 6099. Lecture Notes in Computer Science. Springer, 2010, pp. 89–101.

[31] Peter Burcsi, Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. "On Approximate Jumbled Pattern Matching in Strings". In: *Theory Comput. Syst.* 50.1 (2012), pp. 35–51.

[32] 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.

[33] Paul B. Callahan and S. Rao Kosaraju. "A Decomposition of Multidimensional Point Sets with Applications to k-Nearest-Neighbors and n-Body Potential Fields". In: *J. ACM* 42.1 (1995), pp. 67–90.

[34] Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. "Nondeterministic Extensions of the Strong Exponential Time Hypothesis and Consequences for Non-reducibility". In: *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*. Ed. by Madhu Sudan. ACM, 2016, pp. 261–270.

[35] Coralia Cartis and Andrew Thompson. "An exact tree projection algorithm for wavelets". In: *IEEE Signal Processing Letters* 20.11 (2013), pp. 1026–1029.

[36] Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. "From Gap-ETH to FPT-Inapproximability: Clique, Dominating Set, and More". In: *Proc. 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS'17)*. 2017, pp. 743–754.

[37] Timothy M. Chan. "More algorithms for all-pairs shortest paths in weighted graphs". In: *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*. Ed. by David S. Johnson and Uriel Feige. ACM, 2007, pp. 590–598.

[38] Timothy M. Chan. "Approximation Schemes for 0-1 Knapsack". In: *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*. Ed. by Raimund Seidel. Vol. 61. OASICS. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 5:1–5:12.

[39] Timothy M. Chan and Moshe Lewenstein. "Clustered Integer 3SUM via Additive Combinatorics". In: *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*. STOC '15. Portland, Oregon, USA: ACM, 2015, pp. 31–40.

[40] Lijie Chen. "On The Hardness of Approximate and Exact (Bichromatic) Maximum Inner Product". In: *Proc. 33rd Computational Complexity Conference (CCC'18)*. Vol. 102. 2018, 14:1–14:45.

[41] Fan R. K. Chung, V. Faber, and Thomas A. Manteuffel. "An Upper Bound on the Diameter of a Graph from Eigenvalues Associated with its Laplacian". In: *SIAM J. Discrete Math.* 7.3 (1994), pp. 443–457.

[42] Fan RK Chung. *Spectral graph theory*. Vol. 92. American Mathematical Soc., 1997.

[43] Ferdinando Cicalese, Eduardo Sany Laber, Oren Weimann, and Raphael Yuster. "Approximating the maximum consecutive subsums of a sequence". In: *Theor. Comput. Sci.* 525 (2014), pp. 130–137.

[44] Edith Cohen. "Using Selective Path-Doubling for Parallel Shortest-Path Computations". In: *J. Algorithms* 22.1 (1997), pp. 30–56.

[45] Edith Cohen and Uri Zwick. "All-Pairs Small-Stretch Paths". In: *J. Algorithms* 38.2 (2001), pp. 335–353.

[46] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[47] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dańiel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlstrom. "On Problems As Hard As CNF-SAT". In: *Proceedings of the 2012 IEEE Conference on Computational Complexity (CCC)*. CCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 74–84.

[48] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[49] Marek Cygan, Harold N. Gabow, and Piotr Sankowski. "Algorithmic Applications of Baur-Strassen's Theorem: Shortest Cycles, Diameter, and Matchings". In: *J. ACM* 62.4 (2015), p. 28.

[50] Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. "On Problems Equivalent to (min, +)-Convolution". In: *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*. Ed. by Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl. Vol. 80. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 22:1–22:15.

[51] 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.

[52] Ran Duan and Seth Pettie. "Fast algorithms for (max,min)-matrix multiplication and bottleneck shortest paths". In: *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*. 2009, pp. 384–391.

[53] Ran Duan, Seth Pettie, and Hsin-Hao Su. "Scaling Algorithms for Weighted Matching in General Graphs". In: *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*. 2017, pp. 781–800.

[54] Ran Duan and Hanlin Ren. "Approximating All-Pair Bounded-Leg Shortest Path and APSP-AF in Truly-Subcubic Time". In: *Proc. 45th International Colloquium on Automata, Languages, and Programming (ICALP'18)*. 2018, 42:1–42:12.

[55] David Steven Dummit and Richard M Foote. *Abstract algebra*. Vol. 3. Wiley Hoboken, 2004.

[56] Wayne Eberly. "Black box Frobenius decompositions over small fields". In: *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation, ISSAC 2000, St. Andrews, United Kingdom, August 6-10, 2000*. Ed. by Carlo Traverso. ACM, 2000, pp. 106–113.

[57] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *J. ACM* 19.2 (1972), pp. 248–264.

[58] David Eppstein. "Minimum Range Balanced Cuts via Dynamic Subset Sums". In: *J. Algorithms* 23.2 (1997), pp. 375–385.

[59] Werner Fenchel. "On conjugate convex functions". In: *Canad. J. Math* 1.73-77 (1949).

[60] Robert W. Floyd. "Algorithm 97: Shortest path". In: *Commun. ACM* 5.6 (1962), p. 345.

[61] Gudmund Skovbjerg Frandsen and Piotr Sankowski. "Dynamic normal forms and dynamic characteristic polynomial". In: *International Colloquium on Automata, Languages, and Programming*. Springer Berlin Heidelberg. 2008, pp. 434–446.

[62] Michael L. Fredman. "How Good is the Information Theory Bound in Sorting?" In: *Theor. Comput. Sci.* 1.4 (1976), pp. 355–361.

[63] Michael L. Fredman. "New Bounds on the Complexity of the Shortest Path Problem". In: *SIAM J. Comput.* 5.1 (1976), pp. 83–89.

[64] Linton C Freeman. "Centrality in social networks conceptual clarification". In: *Social networks* 1.3 (1978), pp. 215–239.

[65] Harold N. Gabow and Robert Endre Tarjan. "Faster Scaling Algorithms for Network Problems". In: *SIAM J. Comput.* 18.5 (1989), pp. 1013–1036.

[66] Harold N. Gabow and Robert Endre Tarjan. "Faster Scaling Algorithms for General Graph-Matching Problems". In: *J. ACM* 38.4 (1991), pp. 815–853.

[67] Anka Gajentaan and Mark H. Overmars. "On a Class of $O(n^2)$ Problems in Computational Geometry". In: *Comput. Geom.* 5 (1995), pp. 165–185.

[68] 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.

[69] Zvi Galil and Oded Margalit. "An Almost Linear-Time Algorithm for the Dense Subset-Sum Problem". In: *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Madrid, Spain, July 8-12, 1991, Proceedings*. Ed. by Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo. Vol. 510. Lecture Notes in Computer Science. Springer, 1991, pp. 719–727.

[70] Zvi Galil and Oded Margalit. personal communication. 2017.

[71] F.R. Gantmacher. *The Theory of Matrices, Vol. 1*. Chelsea, 1959.

[72] MR Garey and DS Johnson. "Computers and intractability: a guide to the theory of NP-completeness". In: (1979).

[73] GV Gens and EV Levner. "Approximation algorithm for some scheduling problems". In: *Engrg. Cybernetics* 6 (1978), pp. 38–46.

[74] George Gens and Eugene Levner. "Computational Complexity of Approximation Algorithms for Combinatorial Problems". In: *Mathematical Foundations of Computer Science 1979, Proceedings, 8th Symposium, Olomouc, Czechoslovakia, September 3-7, 1979*. Ed. by Jirí Becvár. Vol. 74. Lecture Notes in Computer Science. Springer, 1979, pp. 292–300.

[75] George Gens and Eugene Levner. "A fast approximation algorithm for the subset-sum problem". In: *INFOR: Information Systems and Operational Research* 32.3 (1994), pp. 143–148.

[76] Georgii V Gens and Eugenii V Levner. "Fast approximation algorithms for knapsack type problems". In: *Optimization Techniques*. Springer, 1980, pp. 185–194.

[77] Beat Gfeller. "Finding Longest Approximate Periodic Patterns". In: *Algorithms and Data Structures - 12th International Symposium, WADS 2011, New York, NY, USA, August 15-17, 2011. Proceedings*. Ed. by Frank Dehne, John Iacono, and Jörg-Rüdiger Sack. Vol. 6844. Lecture Notes in Computer Science. Springer, 2011, pp. 463–474.

[78] Mark Giesbrecht, Michael J. Jacobson Jr., and Arne Storjohann. "Algorithms for Large Integer Matrix Problems". In: *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 14th International Symposium, AAECC-14, Melbourne, Australia November 26-30, 2001, Proceedings*. Ed. by Serdar Boztas and Igor E. Shparlinski. Vol. 2227. Lecture Notes in Computer Science. Springer, 2001, pp. 297–307.

[79] Omer Gold and Micha Sharir. "Improved Bounds for 3SUM, k-SUM, and Linear Degeneracy". In: *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*. Ed. by Kirk Pruhs and Christian Sohler. Vol. 87. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 42:1–42:13.

[80] Andrew V. Goldberg. "Scaling Algorithms for the Shortest Paths Problem". In: *SIAM J. Comput.* 24.3 (1995), pp. 494–504.

[81] Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. "How Hard is it to Find (Honest) Witnesses?" In: *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*. Ed. by Piotr Sankowski and Christos D. Zaroliagis. Vol. 57. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 45:1–45:16.

[82] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3. ed.)* Johns Hopkins University Press, 1996.

[83] Daniel Graf, Karim Labib, and Przemyslaw Uznanski. "Brief Announcement: Hamming Distance Completeness and Sparse Matrix Multiplication". In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*. Ed. by Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 109:1–109:4.

[84] Brian Hayes. "Computing science: The easiest hard problem". In: *American Scientist* 90.2 (2002), pp. 113–117.

[85] Dorit S. Hochbaum. "Lower and Upper Bounds for the Allocation Problem and Other Nonlinear Optimization Problems". In: *Math. Oper. Res.* 19.2 (1994), pp. 390–409.

[86] Ellis Horowitz and Sartaj Sahni. "Computing Partitions with Applications to the Knapsack Problem". In: *J. ACM* 21.2 (1974), pp. 277–292.

[87] Nick Howgrave-Graham and Antoine Joux. "New Generic Algorithms for Hard Knapsacks". In: *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*. Ed. by Henri Gilbert. Vol. 6110. Lecture Notes in Computer Science. Springer, 2010, pp. 235–256.

[88] Chloe Ching-Yun Hsu and Chris Umans. "On Multidimensional and Monotone k-SUM". In: *To appear at MFCS 2017* (2017).

[89] 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.

[90] Russell Impagliazzo and Ramamohan Paturi. "On the Complexity of k-SAT". In: *J. Comput. Syst. Sci.* 62.2 (2001), pp. 367–375.

[91] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. "Which Problems Have Strongly Exponential Complexity?" In: *J. Comput. Syst. Sci.* 63.4 (2001), pp. 512–530.

[92] Klaus Jansen and Stefan Erich Julius Kraft. "A Faster FPTAS for the Unbounded Knapsack Problem". In: *Combinatorial Algorithms - 26th International Workshop, IWOCA 2015, Verona, Italy, October 5-7, 2015, Revised Selected Papers*. Ed. by Zsuzsanna Lipták and William F. Smyth. Vol. 9538. Lecture Notes in Computer Science. Springer, 2015, pp. 274–286.

[93] Ce Jin. "An Improved FPTAS for 0-1 Knapsack". In: *CoRR* abs/1904.09562 (2019). arXiv: 1904.09562.

[94] Edward G. Coffman Jr. and George S. Lueker. *Probabilistic analysis of packing and partitioning algorithms*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1991.

[95] R.M. Karp. "The fast approximate solution to hard combinatorial problems". In: *Proceedings of the 6th Southeastern Conference on Combinatorics, Graph Theory and Computing* (1975), pp. 15–31.

[96] Richard M. Karp. "Reducibility Among Combinatorial Problems". In: *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York*. Ed. by Raymond E. Miller and James W. Thatcher. The IBM Research Symposia Series. Plenum Press, New York, 1972, pp. 85–103.

[97] Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. "On the parameterized complexity of approximating dominating set". In: *Proc. 50th Annual Symposium on Theory of Computing (STOC'18)*. 2018, pp. 1283–1296.

[98] 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.

[99] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

[100] Hans Kellerer, Ulrich Pferschy, and Maria Grazia Speranza. "An Efficient Approximation Scheme for the Subset-Sum Problem". In: *Algorithms and Computation, 8th International Symposium, ISAAC '97, Singapore, December 17-19, 1997, Proceedings*. Ed. by Hon Wai Leong, Hiroshi Imai, and Sanjay Jain. Vol. 1350. Lecture Notes in Computer Science. Springer, 1997, pp. 394–403.

[101] Philip N. Klein and S. Sairam. "A Parallel Randomized Approximation Scheme for Shortest Paths". In: *Proc. 24th Annual ACM Symposium on Theory of Computing (STOC'92)*. 1992, pp. 750–758.

[102] Konstantinos Koiliaris and Chao Xu. "A Faster Pseudopolynomial Time Algorithm for Subset Sum". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '17. Barcelona, Spain: Society for Industrial and Applied Mathematics, 2017, pp. 1062–1072.

[103] S. Rao Kosaraju. "Efficient Tree Pattern Matching (Preliminary Version)". In: *Proc. 30th Annual Symposium on Foundations of Computer Science (FOCS'89)*. 1989, pp. 178–183.

[104] Eduardo Sany Laber, Wilfredo Bardales Roncalla, and Ferdinando Cicalese. "On lower bounds for the Maximum Consecutive Subsums Problem and the (min, +)-convolution". In: *2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014*. IEEE, 2014, pp. 1807–1811.

[105] Eugene L Lawler. "Fast approximation algorithms for knapsack problems". In: *Mathematics of Operations Research* 4.4 (1979), pp. 339–356.

[106] François Le Gall. "Powers of tensors and fast matrix multiplication". In: *Proceedings of the 39th international symposium on symbolic and algebraic computation*. ACM. 2014, pp. 296–303.

[107] François Le Gall and Harumich Nishimura. "Quantum Algorithms for Matrix Products over Semirings". In: *Chicago J. Theor. Comput. Sci.* 2017 (2017).

[108] Arthur Lim and Jialing Dai. "On product of companion matrices". In: *Linear Algebra and its Applications* 435.11 (2011), pp. 2921–2935.

[109] Ohad Lipsky and Ely Porat. "Approximate Pattern Matching with the $L_1$, $L_2$ and $L_\infty$ Metrics". In: *Algorithmica* 60.2 (2011), pp. 335–348.

[110] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. "Known Algorithms on Graphs of Bounded Treewidth Are Probably Optimal". In: *ACM Trans. Algorithms* 14.2 (2018), 13:1–13:30.

[111] Yves Lucet. "Faster than the Fast Legendre Transform, the Linear-time Legendre Transform". In: *Numerical Algorithms* 16.2 (1997), pp. 171–185.

[112] George B Mathews. "On the partition of numbers". In: *Proceedings of the London Mathematical Society* 1.1 (1896), pp. 486–490.

[113] Ralph C. Merkle and Martin E. Hellman. "Hiding information and signatures in trapdoor knapsacks". In: *IEEE Trans. Information Theory* 24.5 (1978), pp. 525–530.

[114] Stephan Mertens. "The easiest hard problem: Number partitioning". In: *Computational Complexity and Statistical Physics* 125.2 (2006), pp. 125–139.

[115] Tanaeem M. Moosa and M. Sohel Rahman. "Indexing permutations for binary strings". In: *Inf. Process. Lett.* 110.18-19 (2010), pp. 795–798.

[116] Marcin Mucha, Jesper Nederlof, Jakub Pawlewicz, and Karol Wegrzycki. "Equal-Subset-Sum Faster Than the Meet-in-the-Middle". In: *ESA'19* (2019).

[117] Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. "A Subquadratic Approximation Scheme for Partition". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*. Ed. by Timothy M. Chan. SIAM, 2019, pp. 70–88.

[118] Thom Mulders and Arne Storjohann. "Rational solutions of singular linear systems". In: *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation, ISSAC 2000, St. Andrews, United Kingdom, August 6-10, 2000*. Ed. by Carlo Traverso. ACM, 2000, pp. 242–249.

[119] Aran Nayebi and Virginia Vassilevska Williams. "Quantum algorithms for shortest paths problems in structured instances". In: *CoRR* (2014). arXiv: 1410.6220.

[120] Jesper Nederlof, Erik Jan van Leeuwen, and Ruben van der Zwaan. "Reducing a Target Interval to a Few Exact Queries". In: *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*. Ed. by Branislav Rovan, Vladimiro Sassone, and Peter Widmayer. Vol. 7464. Lecture Notes in Computer Science. Springer, 2012, pp. 718–727.

[121] Tore Opsahl, Filip Agneessens, and John Skvoretz. "Node centrality in weighted networks: Generalizing degree and shortest paths". In: *Social Networks* 32.3 (2010), pp. 245–251.

[122] James B. Orlin. "A Faster Strongly Polynomial Minimum Cost Flow Algorithm". In: *Operations Research* 41.2 (1993), pp. 338–350.

[123] 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.

[124] Mihai Patrascu. "Towards polynomial lower bounds for dynamic problems". In: *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*. Ed. by Leonard J. Schulman. ACM, 2010, pp. 603–610.

[125] John H. Reif and Stephen R. Tate. "On Dynamic Algorithms for Algebraic Problems". In: *J. Algorithms* 22.2 (1997), pp. 347–371.

[126] Liam Roditty and Asaf Shapira. "All-Pairs Shortest Paths with a Sublinear Additive Error". In: *Proc. 35th International Colloquium on Automata, Languages and Programming (ICALP'08)*. 2008, pp. 622–633.

[127] Liam Roditty and Asaf Shapira. "All-pairs Shortest Paths with a Sublinear Additive Error". In: *ACM Trans. Algorithms* 7.4 (Sept. 2011), 45:1–45:12.

[128] Liam Roditty and Virginia Vassilevska Williams. "Minimum Weight Cycles and Triangles: Equivalences and Algorithms". In: *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*. Ed. by Rafail Ostrovsky. IEEE Computer Society, 2011, pp. 180–189.

[129] Aviad Rubinstein. "Hardness of approximate nearest neighbor search". In: *Proc. 50th Annual Symposium on Theory of Computing (STOC'18)*. 2018, pp. 1260–1268.

[130] Piotr Sankowski and Karol Wegrzycki. "Improved Distance Queries and Cycle Counting by Frobenius Normal Form". In: *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*. Ed. by Heribert Vollmer and Brigitte Vallée. Vol. 66. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 56:1–56:14.

[131] Piotr Sankowski and Karol Węgrzycki. "Improved Distance Queries and Cycle Counting by Frobenius Normal Form". In: *Theory of Computing Systems* (2018).

[132] Ludwig Schmidt. personal communication. 2017.

[133] Arnold Schönhage. "On the power of random access machines". In: *Proc. 6th International Colloquium on Automata, Languages, and Programming (ICALP'79)*. Vol. 71. 1979, pp. 520–529.

[134] Alexander Schrijver. "A Combinatorial Algorithm Minimizing Submodular Functions in Strongly Polynomial Time". In: *J. Comb. Theory, Ser. B* 80.2 (2000), pp. 346–355.

[135] Richard Schroeppel and Adi Shamir. "A T=O($2^{n/2}$), S=O($2^{n/4}$) Algorithm for Certain NP-Complete Problems". In: *SIAM J. Comput.* 10.3 (1981), pp. 456–464.

[136] J. T. Schwartz. "Fast Probabilistic Algorithms for Verification of Polynomial Identities". In: *J. ACM* 27.4 (Oct. 1980), pp. 701–717.

[137] Raimund Seidel. "On the All-Pairs-Shortest-Path Problem". In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*. Ed. by S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis. ACM, 1992, pp. 745–749.

[138] Oliver Serang. "A Fast Numerical Method for Max-Convolution and the Application to Efficient Max-Product Inference in Bayesian Networks". In: *Journal of Computational Biology* 22.8 (2015), pp. 770–783.

[139] Asaf Shapira, Raphael Yuster, and Uri Zwick. "All-pairs bottleneck paths in vertex weighted graphs". In: *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*. 2007, pp. 978–985.

[140] Daniel D. Sleator and Robert Endre Tarjan. "A Data Structure for Dynamic Trees". In: *J. Comput. Syst. Sci.* 26.3 (June 1983), pp. 362–391.

[141] Arne Storjohann. "An $O(n^3)$ Algorithm for the Frobenius Normal Form". In: *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation, ISSAC '98, Rostock, Germany, August 13-15, 1998*. Ed. by Volker Weispfenning and Barry M. Trager. ACM, 1998, pp. 101–105.

[142] Arne Storjohann. "Deterministic Computation of the Frobenius Form". In: *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, 2001, pp. 368–377.

[143] Zhihui Tang, Ramani Duraiswami, and Nail A. Gumerov. "Fast algorithms to compute matrix-vector products for pascal matrices". In: *Technical Reports from UMIACS UMIACS-TR-2004-08* (2004).

[144] Éva Tardos. "A strongly polynomial minimum cost circulation algorithm". In: *Combinatorica* 5.3 (1985), pp. 247–256.

[145] Mikkel Thorup. "Undirected Single-Source Shortest Paths with Positive Integer Weights in Linear Time". In: *J. ACM* 46.3 (1999), pp. 362–394.

[146] Mikkel Thorup. "Floats, Integers, and Single Source Shortest Paths". In: *J. Algorithms* 35.2 (2000), pp. 189–201.

[147] Mikkel Thorup. "Equivalence between Priority Queues and Sorting". In: *Proc. 43rd Symposium on Foundations of Computer Science (FOCS'02)*. 2002, pp. 125–134.

[148] Mikkel Thorup and Uri Zwick. "Approximate distance oracles". In: *J. ACM* 52.1 (2005), pp. 1–24.

[149] Godfried Toussaint. "The Geometry of Musical Rhythm". In: *Discrete and Computational Geometry: Japanese Conference, JCDCG 2004, Tokyo, Japan, October 8-11, 2004, Revised Selected Papers*. Ed. by Jin Akiyama, Mikio Kano, and Xuehou Tan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 198–212.

[150] Carlo Traverso, ed. *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation, ISSAC 2000, St. Andrews, United Kingdom, August 6-10, 2000*. ACM, 2000.

[151] Virginia Vassilevska. "Efficient algorithms for path problems in weighted graphs". In: *Carnegie Mellon University* (2008). PhD Thesis.

[152] Virginia Vassilevska and Ryan Williams. "Finding a maximum weight triangle in $O(n^{3-\delta})$ time, with applications". In: *Proc. 38th Annual ACM Symposium on Theory of Computing (STOC'06)*. 2006, pp. 225–231.

[153] Virginia Vassilevska, Ryan Williams, and Raphael Yuster. "All Pairs Bottleneck Paths and Max-Min Matrix Products in Truly Subcubic Time". In: *Theory of Computing* 5.1 (2009), pp. 173–189.

[154] Virginia Vassilevska and Williams Ryan Williams. "Subcubic equivalences between path, matrix and triangle problems". In: *In Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS)*. 2010.

[155] László A. Végh. "A Strongly Polynomial Algorithm for Generalized Flow Maximization". In: *Math. Oper. Res.* 42.1 (2017), pp. 179–211.

[156] Stephen Warshall. "A Theorem on Boolean Matrices". In: *J. ACM* 9.1 (1962), pp. 11–12.

[157] R. Ryan Williams. "Faster All-Pairs Shortest Paths via Circuit Complexity". In: *SIAM J. Comput.* 47.5 (2018), pp. 1965–1985.

[158] Ryan Williams. "A new algorithm for optimal 2-constraint satisfaction and its implications". In: *Theor. Comput. Sci.* 348.2-3 (2005), pp. 357–365.

[159] Virginia Vassilevska Williams. "Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis (Invited Talk)". In: *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*. Ed. by Thore Husfeldt and Iyad A. Kanj. Vol. 43. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 17–29.

[160] 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.

[161] Virginia Vassilevska Williams and Ryan Williams. "Finding, minimizing, and counting weighted subgraphs". In: *SIAM Journal on Computing* 42.3 (2013), pp. 831–854.

[162] Gerhard J. Woeginger. "When Does a Dynamic Programming Formulation Guarantee the Existence of a Fully Polynomial Time Approximation Scheme (FPTAS)?" In: *INFORMS Journal on Computing* 12.1 (2000), pp. 57–74.

[163] Raphael Yuster. "Efficient algorithms on sets of permutations, dominance, and real-weighted APSP". In: *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*. 2009, pp. 950–957.

[164] Raphael Yuster. "A shortest cycle for each vertex of a graph". In: *Inf. Process. Lett.* 111.21-22 (2011), pp. 1057–1061.

[165] Raphael Yuster. "Approximate shortest paths in weighted graphs". In: *J. Comput. Syst. Sci.* 78.2 (2012), pp. 632–637.

[166] Raphael Yuster and Uri Zwick. "Finding Even Cycles Even Faster". In: *Automata, Languages and Programming, 21st International Colloquium, ICALP94, Jerusalem, Israel, July 11-14, 1994, Proceedings*. Ed. by Serge Abiteboul and Eli Shamir. Vol. 820. Lecture Notes in Computer Science. Springer, 1994, pp. 532–543.

[167] Raphael Yuster and Uri Zwick. "Answering distance queries in directed graphs using fast matrix multiplication". In: *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*. IEEE Computer Society, 2005, pp. 389–396.

[168] Richard Zippel. "Probabilistic algorithms for sparse polynomials". In: *Symbolic and Algebraic Computation*. Ed. by Edward W. Ng. Berlin, Heidelberg: Springer Berlin Heidelberg, 1979, pp. 216–226.

[169] Uri Zwick. "All Pairs Shortest Paths in Weighted Directed Graphs – Exact and Almost Exact Algorithms". In: *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*. IEEE Computer Society, 1998, pp. 310–319.

[170] Uri Zwick. "All pairs shortest paths using bridging sets and rectangular matrix multiplication". In: *J. ACM* 49.3 (2002), pp. 289–317.