Dynamic Programming meets Fine-grained Complexity

by

Xiao Mao

B.S. Computer Science and Engineering and Mathematics
Massachusetts Institute of Technology, 2021

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

© Massachusetts Institute of Technology 2022. All rights reserved.

Author . . . .
Department of Electrical Engineering and Computer Science
August 5, 2022

Certified by . . . .
Virginia Vassilevska Williams
Professor
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Dynamic Programming meets Fine-grained Complexity

by

Xiao Mao

Submitted to the Department of Electrical Engineering and Computer Science
on August 5, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Since the term was coined by Richard Bellman in the 1940s, Dynamic Programming (DP) has remained one of the most popular technique in theoretical computer science, and has found applications in a wide range of problems.

In this thesis, I summarize my three recent works covering applications of DP to three fundamental problems in fine-grained complexity. The first application is a sub-cubic time algorithm for unweighted tree edit distance (TED), the second application is an improved FPTAS (Fully Polynomial-Time Approximation Scheme) for Partition, and the third application is an improved FPTAS for Knapsack.

Thesis Supervisor: Virginia Vassilevska Williams
Title: Professor

Acknowledgments

I would like to thank Professor Virginia Vassilevska Williams for supervising this thesis.

I would like to thank Ce Jin and Mingyang Deng for helping with my write-up.

I would like to thank my parents for immersing me in mathematics and computer science at an early age. The competitive programming experience during my middle and high school years was instrumental in my success in research so far. Therefore, I would also like to thank everyone who had supported me during my competitive programming career.

Last but not least, I would like to thank all other people who have supported my career in academics.

Contents

1 Introduction **13**

2 Algorithms for Tree Edit Distance **15**

2.1 Introduction and Preliminaries 15

2.1.1 Introduction 15

2.1.2 Tree edit distance related definitions 16

2.1.3 A computation model for row-monotone, column-monotone matrices . 22

2.2 Our algorithm 23

2.2.1 A novel cubic algorithm 23

2.2.2 Main algorithm 27

2.2.3 Transition between synchronous subforests 30

3 Faster Approximation Schemes for Partition and Knapsack **37**

3.1 Introduction 37

3.1.1 Background 37

3.1.2 Our results 39

3.2 Preliminaries 39

3.2.1 Problem Statements 39

3.2.2 Sumsets and Subset Sums 40

3.2.3 Knapsack Problem and Profit functions 40

3.2.4 $(1-\delta, \Delta)$ approximation up to t 41

3.2.5 Implicit Dynamic Progamming Schemes 43

3.2.6 Additive Combinatorics . . . 43
3.3 Approximating Partition . . . 44
3.4 Approximating Knapsack . . . 51
3.4.1 Known lemmas . . . 51
3.4.2 Reduction Based on Greedy Exchange Argument . . . 52
3.4.3 Approximation using Δ-multiples of small set Δ . . . 57
3.4.4 Random Partitioning . . . 58

4 Concluding Remarks 63

List of Figures

2-1 An optimal series of operations to make T_1 and T_2 identical is shown, and $\mathsf{ed}(T_1, T_2) = 3$. . . . 18
2-2 The bi-order traversal sequence and a synchronous subforest of tree T 20
2-3 The mapping of maximum weight between T_1 and T_2 21
2-4 Transition between synchronous subforests 32

List of Tables

Chapter 1

Introduction

The term “Dynamic Programming” (DP) was coined by Richard Bellman in the 1940s. “Programming” in the modern sense had not developed and the word was in its original sense of ”planning.” Dynamic Programming was therefore planning for a task where decisions need to be made one after another, hence the adjective “dynamic.”

Today, this nearly century-old technique has remained one of the most popular and most important technique in theoretical computer science, and is covered in nearly all fundamental college-level algorithm classes. It has found applications in a wide range of problems, and new variants and extensions of many of those applications, including the most well-known ones, are still being discovered. Two of the most well-known applications are the Longest-Common-Subsequence (LCS) problem and the Knapsack problem, to which DP-based solutions have been known at least since the 70s. The new algorithms introduced in this thesis solve variants of these two problems.

There are often no good alternative techniques to DP, and thus new algorithms for problems with known DP-based solutions often still have to rely on a DP scheme, novel or traditional. This DP scheme can often be combined with other techniques for better efficiency. Firstly, the DP schemes themselves can sometimes be sped up using some advanced sub-routines. Secondly, new techniques can often be built upon the DP scheme to prune useless computation or to re-order different stages of computation for better efficiency. In

this thesis, we will see combinatorial and number theoretical approaches built upon DP schemes, and the DP schemes themselves are sped up using fast Matrix Multiplication and Fast Fourier Transforms.

In this thesis, I will summarize my work showing improved fine-grained upper bounds for many fundamental problems. I will show an improved exact algorithm the Unweighted Tree Edit Distance problem in Chapter 2, and improved Fully-Polynomial-Time-Approximation-Schemes (FPTASes) for the Partition problem and the Knapsack problem in Chapter 3.

Chapter 2

Algorithms for Tree Edit Distance

© 2021 IEEE. This chapter is reprinted, with permission from IEEE, from my paper *Breaking the Cubic Barrier for (Unweighted) Tree Edit Distance*, published in the proceedings of *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*.

2.1 Introduction and Preliminaries

2.1.1 Introduction

One of the most fundamental problems in computer science is the *(string) edit distance* problem, studied since the 1960's. Defined as the minimum number of deletions, insertions or substitutions needed to change one string into another, it is a natural way to measure the dissimilarity between data that can be represented as strings. As a measure for linearly ordered data, edit distance is not as useful when the data is hierarchically organized. For data that is in the form of ordered trees, *tree edit distance* serves as a natural generalization of edit distance. First introduced by Selkow in 1977 [41], it has found applications in a variety of areas such as computational biology [21, 42, 23, 44], structured data analysis [10, 13, 18], image processing [3, 33, 32, 40], and compiler optimization [17]. One of the most notable applications is the analysis of RNA molecules whose secondary structures are typically represented as rooted trees [21, 22].

For two *rooted ordered* trees whose nodes are labeled with symbols, their tree edit distance is the minimum number of node deletions, insertions, and relabelings needed to change one tree into the other. When a node is deleted, its children become children of its parent. The resulting trees must be structurally identical, where the order of siblings matters, and symbols on corresponding nodes must also match. A formal definition will be given in the next sub-section.

The main result we will achieve in this chapter is the following:

Theorem 2.1.1. *There is an $O(nm^{1.9546})$ time randomized algorithm and an $O(nm^{1.9639})$ time deterministic algorithm that computes the (unweighted) tree edit distance between two trees of sizes n and m.*

When $m = O(n)$, this implies an $O(n^{2.9546})$ time randomized and an $O(n^{2.9639})$ time deterministic algorithm that solve the tree edit distance problem, which are the first ever known truly sub-cubic algorithms for this problem.

The two exponents in our results are from applications of the work by Bringmann, Grandoni, Saha, and Vassilevska Williams [7], which shows that the max-plus product of two *bounded-difference* $n \times n$ matrices (i.e. bounded difference between adjacent entries, full definition given in the original paper) can be computed in $O(n^{2.8244})$ randomized and $O(n^{2.8603})$ deterministic time. These were the optimal algorithms for this problem at the time I wrote the original FOCS paper. At the time the thesis is completed, better algorithms have been found [15] and the running times in Theorem 2.1.1 can be improved trivially. We will stay faithful to the original paper and use the old running times.

We will first look at a novel (nearly-)cubic algorithm for the tree edit distance problem. Such running time had been achieved before the publication of my work [31, 17], but our algorithm stood out in that its ideas were extended to give a sub-cubic running time.

2.1.2 Tree edit distance related definitions

The tree edit distance problem involves ordered trees. For an ordered tree T, each node of T is labeled with a symbol from some given alphabet Σ. In this paper, we consider the size

of the alphabet to be $O(n)$. For a node $u \in T$, let $\mathsf{par}(u)$ denote the parent node of u. Let $\mathsf{root}(T)$ be the root of the tree T.

We treat a forest F as ordered as well, meaning that the order between the trees in the forests is important. Under this setting, we can treat a forest F as a tree with a *virtual root*, and let $\mathsf{par}(v)$ be the virtual root if v is the root of a tree in the forest. Let $\mathsf{sub}(u)$ be the subtree of u. We let L_F denote the leftmost tree in F and R_F denote the rightmost tree in F. For a tree $T \in F$, let $F - T$ be the forest we get by removing the tree T from F while keeping the order of the remaining trees. Let $|F|$ be the number of nodes in F. For a sequence of forests $F_1, F_2, \cdots, F_k$, let $F_1 + F_2 + \cdots + F_k$ be the concatenation of these forests from left to right. For two forests F_1, F_2 of F, we say $F_1 \subset F_2$ if all nodes in F_1 are in F_2. When $F_1 \subset F_2$, we use $F_2 \backslash F_1$ to denote the set of nodes in F_2 that are not in F_1. An empty forest is denoted by $\emptyset$.

For a node u in a forest F, let $\mathsf{d}(u)$ be the number of children of u. For $1 \leq k \leq \mathsf{d}(u)$, let $\mathsf{child}(u, k)$ be the k-th child of u from left to right. Let $\mathsf{sub}(u, x) = \mathsf{sub}(\mathsf{child}(u, x))$ and $\mathsf{sub}(u, [x, y]) = \mathsf{sub}(u, x) + \mathsf{sub}(u, x + 1) + \cdots + \mathsf{sub}(u, y)$. Specifically, for the virtual root r of the forest F, let $\mathsf{d}(r)$ be the number of trees in F and $\mathsf{child}(r, k)$ be the root of the k-th tree from left to right, and define $\mathsf{sub}(r, x)$ and $\mathsf{sub}(r, [x, y])$ similarly.

The *node removal operation* removes a node v from the forest F, and let the children of v become children of $\mathsf{par}(v)$, with the same ordering. The result of the removal is denoted by $F - v$.

Definition 2.1.2 ((Unweighted) Tree Edit Distance)**.** For two forests F_1 and F_2, we consider the following two types of operations:

- Relabeling: changing the label of a node to another symbol in Σ.
- Deletion: using the node removal operation to remove a node.

The *tree edit distance* between F_1 and F_2, denoted by $\mathsf{ed}(F_1, F_2)$, is the minimum number of operations we can perform on F_1 and F_2 so that they become identical forests.

Some literature distinguishes between *forest edit distance* and tree edit distance, but for simplicity we will not make this distinction.

Figure 2-1 gives an example of tree edit distance between two trees T_1 and T_2.

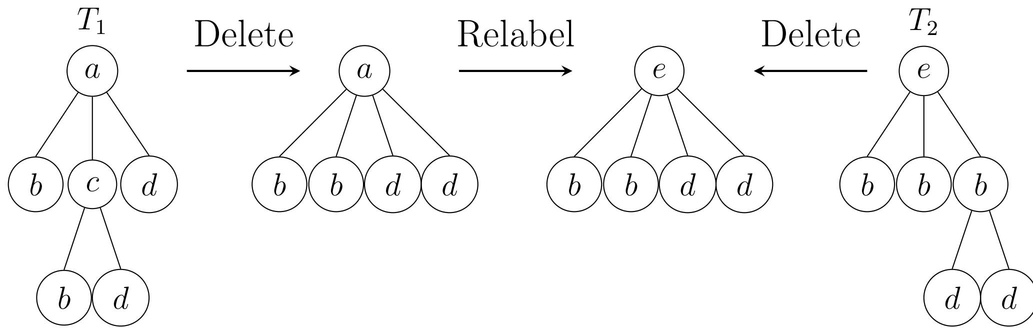


Figure 2-1: An optimal series of operations to make T_1 and T_2 identical is shown, and $\mathsf{ed}(T_1, T_2) = 3$.

We now consider an equivalent maximization problem defined on similarity, which uniquely determines the edit distance:

Definition 2.1.3 (Similarity)**.** The *similarity* between two forests F_1 and F_2 is defined as $\mathsf{sim}(F_1, F_2) = |F_1| + |F_2| - \mathsf{ed}(F_1, F_2)$.

Since it is obvious that $\mathsf{ed}(F_1, F_2) \leq |F_1| + |F_2|$, similarity is always non-negative.

Definition 2.1.4 (Bi-order traversal sequence)**.** Consider the depth-first traversal of a forest F starting from the virtual root, with subtrees recursively traversed from left to right. From that we can generate an *bi-order traversal sequence* of length $2|F|$, where each node appears twice, in the following way:

- Start from the empty sequence.
- Every time we enter or leave a node, we attach the node to the end of the sequence (do not attach the virtual root).

We use $F(i)$ to denote the i-th node in such sequence.

Definition 2.1.5 (Subforest)**.** For $1 \leq l \leq r \leq 2|F|+1$, we use $F[l, r)$ to denote the forest obtained by removing from F all nodes that appear at least once in $F(1), F(2), \cdots, F(l-1)$ or $F(r), F(r+1) \cdots F(2|F|)$, and we call such forest a *subforest* of F (as later illustrated in Figure 2-2).

For a node u, $\mathsf{l}(u)$ equals the first index where u appears in the bi-order traversal sequence and $\mathsf{r}(u)$ equals *one plus* the second index where u appears in the sequence. By these definitions, we can see that $F[l, r)$ contains a node u if and only if $l \leq \mathsf{l}(u)$ and $\mathsf{r}(u) \leq r$ and that $F[\mathsf{l}(u), \mathsf{r}(u))$ is equal to $\mathsf{sub}(u)$. Note that $F[l, r)$ and $F[l', r')$ might be the same forest for distinct pairs (l, r) and (l', r').

We now introduce the definition of "synchronous subforests," named in a similar spirit to the term "synchronous decomposition" from the recent $(1+\varepsilon)$-approximation paper [5], which decomposes the tree to disconnected components that become connected after adding one more node and its incident edges.

Definition 2.1.6 (Synchronous subforest)**.** For a forest F, a subforest F' of F is a *synchronous subforest* of F if there exists a node u that is either a node in F or the virtual root of F, and $1 \leq x \leq y \leq \mathsf{d}(u)$ such that $F' = \mathsf{sub}(u, [x, y])$.

We can see that the node u works as the *virtual root* of F', and we denote u as $\mathsf{vroot}(F')$. Figure 2-2 shows the bi-order traversal sequence of tree T. For node 4, we have $\mathsf{l}(4) = 5$ and $\mathsf{r}(4) = 11$. It also shows a synchronous subforest $F = T[3, 15)$ whose virtual root is node 3. Note that node 2 and 3 are not in $F(3, 15)$ since one of their occurrences is not inside the highlighted interval. We can see that this definition is not a one-on-one mapping, since for example $F(5, 15)$ would be identical to $F(3, 15)$.

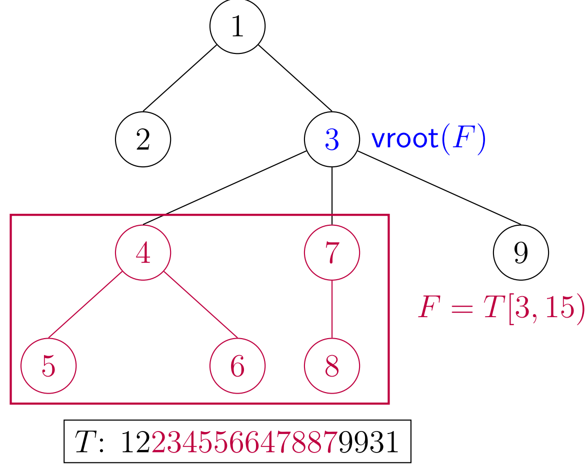


Figure 2-2: The bi-order traversal sequence and a synchronous subforest of tree T

For two nodes u and v labeled with symbols, let $\delta(u, v)$ be equal to 0 if u and v have the same symbols and 1 if their symbols differ. Let $\eta(u, v) = 2 - \delta(u, v)$.

For two nodes u and v such that neither is the ancestor of the other, we have either $\mathsf{r}(u) \leq \mathsf{l}(v)$ or $\mathsf{r}(v) \leq \mathsf{l}(u)$, and we say u precedes v if $\mathsf{r}(u) \leq \mathsf{l}(v)$. For example, node 5 precedes node 6 in the tree T in Figure 2-2.

Mapping The maximization of the similarity between F_1 and F_2 can be interpreted as finding a mapping of maximum weight. The mapping we use is identical to the mapping used in Section 2.2 of [46], except for the fact that we are considering similarity.

The mapping is between two sequences of distinct nodes $\{u_1, u_2, \cdots, u_k\} \in V(F_1)^k$, $\{v_1, v_2, \cdots, v_k\} \in V(F_2)^k$, such that for all $1 \leq i < j \leq k$,

- u_i is an ancestor of u_j in T_1 if and only if v_i is an ancestor of v_j in T_2,
- u_j is an ancestor of u_i in T_1 if and only if v_j is an ancestor of v_i in T_2, and
- If neither of u_i and u_j is the ancestor of the other, u_i precedes u_j in T_1 if and only if v_i precedes v_j in T_2.

For each i we map u_i to v_i, and the weight of the mapping is

$$\sum_{1 \le i \le k} \eta(u_i, v_i).$$

A node $u \in T_1$ is *mapped* if $u \in \{u_1, u_2, \cdots u_k\}$.

Figure 2-3 shows the mapping of maximum weight between the same T_1 and T_2 as in Figure 2-1. Nodes with the same subscripts are mapped to each other. We have $\mathsf{sim}(T_1, T_2) = |T_1| + |T_2| - \mathsf{ed}(T_1, T_2) = 6 + 6 - 3 = 9$, and the mapping indeed has weight 9 since the first pair contributes 1 to the weight and the remaining 4 pairs contribute 2 to the weight.

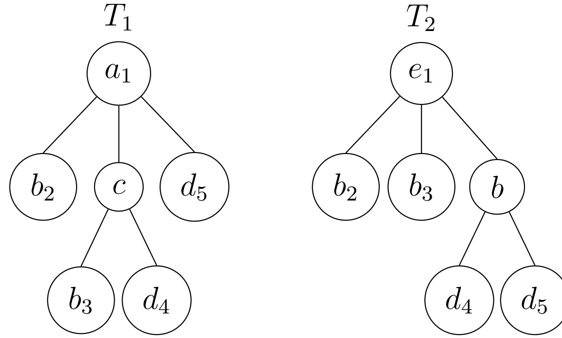


Figure 2-3: The mapping of maximum weight between T_1 and T_2

Definition 2.1.7 (Similarity matrix)**.** For two forests F_1 and F_2, let the *similarity matrix* $S(F_1, F_2) = s_{i,j}(F_1)$ be a $(2|F_2| + 1) \times (2|F_2| + 1)$ matrix where

$$s_{i,j}(F_1) = \begin{cases} \mathsf{sim}(F_1, F_2[i,j]) & \text{if } i \le j \\ -\infty & \text{if } i > j \end{cases}.$$

For two compatible matrices A and B, we use $A \star B$ to denote the *max-plus product* of A and B, which is a matrix $C = c_{ij}$ where $c_{ij} = \max_k \{a_{ik} + b_{kj}\}$.

An $n \times m$ matrix $A = a_{ij}$ is called *row-monotone* if $a_{i,j} \le a_{i,j+1}$ for all i, j, and *column-monotone* if $a_{i+1,j} \le a_{i,j}$ for all i, j.

An $n \times n$ matrix A is called *finite-upper-triangular* if the entries below the main diagonal are $-\infty$ and the entries elsewhere are finite, and is called *W-bounded-upper-triangular* if A

is finite-upper-triangular and the non-$(-\infty)$ entries of A are integers between 0 and W.

2.1.3 A computation model for row-monotone, column-monotone matrices

To speed up matrix manipulations, we store and modify our matrices in an implicit way. We now define a computation model for the matrices involved in our algorithm.

We consider the following *range operations* for $n \times m$ row-monotone, column-monotone matrices. Given an $n \times m$ matrix $A = a_{ij}$, we can

- Produce a new $n \times m$ matrix $B = [-\infty]_{n,m}$ (i.e. entries of B are all $-\infty$),
- Produce a new matrix $B = A$, or
- Given i', j' and $x \in \mathbb{N} \cup \{-\infty\}$, produce a new matrix $B = b_{ij}$ such that

$$b_{ij} = \begin{cases} \max(a_{ij}, x) & \text{if } i \leq i' \text{ and } j' \leq j \\ a_{ij} & \text{otherwise} \end{cases}$$

  and denote such matrix as $B = \mathsf{rangemax}(A, i', j', x)$.

We also consider the following *range queries* on an $n \times m$ row-monotone, column-monotone matrix A:

- Given i, j, query A_{ij},
- Given i, x, query $\mathsf{mincol}(A, i, x) = \min\{j \mid A_{ij} \geq x\}$ or any index in $[1, m]$ if such j does not exist, or
- Given j, x, query $\mathsf{maxrow}(A, j, x) = \max\{i \mid A_{ij} \geq x\}$ or any index in $[1, n]$ if such i does not exist.

There are well-known data structures (e.g. *persistent 2D segment trees*) that can perform the mentioned range operations and range queries in $\widetilde{O}(1)$[1] time assuming that all matrices involved are created using our model.

2.2 Our algorithm

In this section we introduce our algorithm in detail. In Section 2.2.1 we introduce the dynamic programming scheme we start with and a novel cubic algorithm based on this scheme, and motivate the decomposition scheme that we will use to break the cubic barrier. Section 2.2.2 introduces our main algorithm based on that decomposition scheme, which involves two different types of transitions. Section 2.2.3 introduces how the type II transitions between two synchronous subforests with small difference in sizes can be done. The type I transitions which require more efficient computation of max-plus products can be done via a reduction to max-plus product between bounded-difference matrices. The definitions and other details are shown in the original paper and are omitted in this thesis since they are irrelevant to dynamic programming.

We will assume that the input forests are two trees T_1 and T_2. It is easy to extend our algorithm to generic forests. We also assume that $|T_1| \geq |T_2|$. We will use the shorthand $S(F) = S(F, T_2)$ for any forest F since the second argument will always be T_2 in our algorithm.

2.2.1 A novel cubic algorithm

Our dynamic programming scheme

We use a dynamic programming scheme that can be shown to be closely related to the one used in Chen's 2001 algorithm [14], but is fundamentally different from the one used in Zhang and Shasha's algorithm [46] and its descendants [31, 17].

Given a forest F, we use the following dynamic programming scheme to compute $S(F)$:

[1]Throughout this thesis, we use $\widetilde{O}(f)$ to denote $O(f \cdot \operatorname{poly}\log(f))$.

- Basic case: $S(\emptyset)$ is a $(2|T_2|+1) \times (2|T_2|+1)$ finite-upper-triangular matrix whose entries on or above the main diagonal are all zero.

- If F contains one tree, let $u = \mathsf{root}(F)$ and we recurse with

$$s_{i,j}(F) = \max \begin{cases} s_{i,j}(F-u) \\ \max\limits_{v \in T_2[i,j)} \{s_{L,R}(F-u) + \eta(u,v)\} \end{cases}, \tag{2.1}$$

  where $(L,R) = (\mathsf{l}(v)+1, \mathsf{r}(v)-1)$.

- Otherwise, we recurse with

$$S(F) = S(F - R_F) \star S(R_F). \tag{2.2}$$

One can see that to compute $S(T_1)$, our dynamic programming scheme recursively computes exactly $S(\mathsf{sub}(u))$ and $S(\mathsf{sub}(u,[1,k]))$ for all $u \in T_1, 2 \leq k \leq \mathsf{d}(u)$, which gives us $O(|T_1|)$ similarity matrices in total.

Since we need to compute max-plus products between $(2|T_2|+1) \times (2|T_2|+1)$ matrices $O(|T_1|)$ times, the total running time is $\widetilde{O}(|T_1||T_2|^3)$. To show that our new dynamic programming approach is promising, we now show how to improve the running time to cubic by exploiting the properties of the matrices using simple combinatorial methods.

Properties of similarity matrices

We note that for any forest F, $S(F)$ is row-monotone and column-monotone: for $i' \leq i \leq j \leq j'$, $T_2[i,j) \subset T_2[i',j')$ and a mapping from F to $T_2[i,j)$ is also a mapping from $T_2[i',j')$. We also note that $S(F)$ is $2\min(|F|,|T_2|)$-bounded-upper-triangular: there are at most $\min(|F|,|T_2|)$ mapped pairs in a mapping between F and T_2 and each pair only contributes at most 2 to the answer. The original paper defines the "bounded-difference" property for matrices and shows that the similarity matrices admit such property.

Optimization to cubic

We first prove the following theorem, where we are using the range operation/query model from Section 2.1.3.

Theorem 2.2.1. *For a forest F, $S(F)$ can be computed in $\widetilde{O}(|F|^2|T_2|)$ time.*

Note the apparently nonsensical running time: if $|F| = o(|T_2|^{0.5})$, then $S(F)$ can be computed in $o(|T_2|^2)$ time, smaller than the number of entries in $S(F)$. This is because we are using the model in Section 2.1.3, and the running time here means that we can answer all the defined range queries on $S(F)$ after some $o(|T_2|^2)$ time pre-processing.

In order to prove Theorem 2.2.1, we will need to show the following:

Lemma 2.2.2. *Let A, B be $n \times n$ row-monotone, column-monotone matrices. If A is m_A-bounded-upper-triangular and B is m_B-bounded-upper-triangular, then $C = A \star B$ can be computed in $\widetilde{O}(m_A m_B n)$ time.*

Proof of Theorem 2.2.1 from Lemma 2.2.2. Consider applying Lemma 2.2.2 to our dynamic programming scheme. Every time we multiply the similarity matrices of two subforests F_1 and F_2, we contribute $\widetilde{O}(|F_1||F_2||T_2|)$ to the total running time. We know the sum of $|F_1||F_2|$ across all multiplications is $O(|F|^2)$ from a classic argument: $|F_1||F_2|$ is the number of pairs of nodes (x, y) where $x \in F_1$ and $y \in F_2$ and each (x, y) pair will only be counted once. Therefore the total running time is $\widetilde{O}(|F|^2|T_2|)$. □

This proof also implies the following corollary, which will be used again in our main truly sub-cubic algorithm:

Corollary 2.2.3. *Let $\{F_1, F_2, \cdots, F_k\}$ be a sequence of forests. For $1 \le l \le k$, let $G_l = F_1 + F_2 + \cdots + F_l$ be the l-th prefix sum. Then the sequence of similarity matrices of prefix sums $\{S(G_1), S(G_2) \cdots S(G_k)\}$ can be computed in $\widetilde{O}(|G_k|^2|T_2|)$ time. The same result holds for suffix sums.*

To prove Lemma 2.2.2, Algorithm 1 computes $C = A \star B$ in $\widetilde{O}(m_A m_B n)$ time. To show

Algorithm 1 Computation of $C = A \star B$ in Lemma 2.2.2

```
1:  procedure MUL1(A, B)
2:      C ← [−∞]_{n,n}
3:      for j ∈ [1, n] do
4:          for x ∈ [0, m_B] do
5:              for y ∈ [0, m_A] do
6:                  k ← maxrow(B, j, x)
7:                  i ← maxrow(A, k, y)
8:                  C ← rangemax(C, i, j, A_{i,k} + B_{k,j})
9:              end for
10:         end for
11:     end for
12: end procedure
```

its correctness, a triple (i, k, j) is *useful* if $A_{i,k} + B_{k,j} \geq C_{i,j}$ and $\max(A_{i+1,k} + B_{k,j}, A_{i,k+1} + B_{k+1,j}) < C_{i,j}$. A triple (i, k, j) is *covered* by the algorithm if $\mathsf{rangemax}(C, i, j, A_{i,k} + B_{k,j})$ has been produced on line 8. It suffices to show that all useful triples are covered. We can see that a triple (i, k, j) cannot be useful when

- $A_{i,k} = A_{i+1,k}$, since if $A_{i,k} + B_{k,j} \geq C_{i,j}$, then $A_{i+1,k} + B_{k,j} = A_{i,k} + B_{k,j} \geq C_{i,j}$, or
- $B_{k,j} = B_{k+1,j}$, since if $A_{i,k} + B_{k,j} \geq C_{i,j}$, then $A_{i,k+1} + B_{k+1,j} \geq A_{i,k} + B_{k+1,j} = A_{i,k} + B_{k,j} \geq C_{i,j}$.

Therefore, $A_{i,k}$ must be the last occurrence of its value on column k of A and $B_{k,j}$ must be the last occurrence of its value on column j of B. We can now see that the algorithm indeed covers all useful triples.

Note that in our dynamic programming scheme we also need to deal with the case of Equation 2.1. This can easily be done in $\widetilde{O}(|T_2|)$ time by first initializing $S(F)$ with $S(F - u)$ and then enumerating the $O(|T_2|)$ nodes $v \in T_2[i, j)$ and replacing $S(F)$ with $\mathsf{rangemax}(S(F), l(v), r(v), s_{\mathsf{l}(v)+1,\mathsf{r}(v)-1}(F - u) + \eta(u, v))$.

Finally, it seems that the logarithmic factors in our cubic algorithm can be removed by using some more time-efficient way to store the matrices (e.g. by maintaining two 2D tables for a similarity matrix A storing the values of $\mathsf{mincol}(A, i, x)$ and $\mathsf{maxrow}(A, j, x)$). We do not give the full details here since we focus on polynomial improvements.

2.2.2 Main algorithm

Overview

Our main algorithm is based on the following decomposition scheme: for a block size Δ, we decompose the computation of $S(T_1)$ into $O(|T_1|/\Delta)$ transitions between synchronous subforests (Definition 2.1.6) of the following two types:

- Type I: transition from two synchronous subforests F_1 and F_2 to F_1+F_2, where F_1+F_2 is also a synchronous subforest and both $|F_1|$ and $|F_2|$ are no less than Δ.
- Type II: transition from synchronous forest F_1 to synchronous forest F_2 such that $F_1 \subset F_2$ and $|F_2| - |F_1| = O(\Delta)$.

For the type I transitions, by using a reduction to bounded-difference matrix max-plus products, the original paper shows that

Theorem 2.2.4. *For any two subforests F_1 and F_2, $S(F_1) \star S(F_2)$ can be computed in $\mathsf{MUL}(\min(|F_1|, |F_2|, |T_2|), |T_2|)$ time, where $\mathsf{MUL}(m, n) = \widetilde{O}(m^{0.9038}n^2)$ randomized time and $\mathsf{MUL}(m, n) = \widetilde{O}(m^{0.9250}n^2)$ deterministic time.*

For the type II transitions, in Section 2.2.3 we will show the following:

Theorem 2.2.5. *Let F be a forest and let F' be a synchronous subforest of F. If $S(F')$ is known, then $S(F)$ can be computed in $\widetilde{O}(\mathsf{MUL}(|F| - |F'|, |T_2|) + |T_2|(|F| - |F'|)^3)$ time, where $\mathsf{MUL}(|F| - |F'|, |T_2|)$ refers to the running time in Theorem 2.2.4.*

In Section 2.2.2 we give our implementation of the decomposition scheme, and in Section 2.2.2 we analyze its running time using Theorem 2.2.4 and Theorem 2.2.5.

Implementation

Algorithm 2 implements the decomposition scheme. For a synchronous subforest F, if F contains more than one trees and both $|L_F|$ and $|R_F|$ are no less than Δ, we compute $S(F)$ using a type I transition from L_F and $F - L_F$. Otherwise, we find a synchronous subforest

F' of F such that $|F| - |F'|$ is $O(\Delta)$ and use a type II transition from F' to F. To do this, if $|F| \leq 3\Delta$ we let $F' = \emptyset$. Otherwise we let $F' = F$, keep removing some part from F' and stop when the next removal will result in $|F| - |F'|$ being greater than 2Δ.

Algorithm 2 Computation of $S(F)$ by decomposition

```
1: procedure COMPUTE(F)
2:     if L_F ≠ R_F AND |L_F| ≥ Δ AND |R_F| ≥ Δ then                ▷ Type I
3:         COMPUTE(L_F)
4:         COMPUTE(F − L_F)
5:         Compute S(F) = S(L_F) ⋆ S(F − L_F) using Theorem 2.2.4
6:     else                                                          ▷ Type II
7:         if |F| ≤ 3Δ then
8:             F' ← ∅
9:         else
10:            F' ← F
11:            while TRUE do
12:                F_NEXT ← F'
13:                if F' contains only one tree then
14:                    F_NEXT ← F' − root(F')
15:                else
16:                    if |L_F'| < |R_F'| then
17:                        F_NEXT ← F' − L_F'
18:                    else
19:                        F_NEXT ← F' − R_F'
20:                    end if
21:                end if
22:                if |F| − |F_NEXT| > 2Δ then
23:                    break
24:                end if
25:                F' ← F_NEXT
26:            end while
27:            COMPUTE(F')
28:        end if
29:        Compute S(F) from S(F') using Theorem 2.2.5
30:    end if
31: end procedure
```

We now show that the total amount of transitions is indeed $O(|T_1|/\Delta)$. Note that each time we have a type I transition, we merge two subforests both of sizes no less than Δ, so the total number of such transitions is $O(|T_1|/\Delta)$. For the type II transitions, we further

divide them into two cases:

- (The first case) F' contains less than two trees or one of $|L_{F'}|$ and $|R_{F'}|$ is less than Δ, or
- (The second case) F' contains at least two trees and both $|L_{F'}|$ and $|R_{F'}|$ are no less than Δ.

For the first case, we have $|F| - |F'| > \Delta$ since otherwise we would not have stopped the removal process. Note that $F \backslash F'$ is disjoint across different type II transitions. Therefore there are no more than $|T_1|/(\Delta + 1) = O(|T_1|/\Delta)$ type II transitions of the first case. For the second case, we can see that the transition we will use to compute $S(F')$ in the next recursive call will be type I . Therefore, the total number of type II transitions of the second case is bounded by the total number of type I transitions, which as we have already shown is $O(|T_1|/\Delta)$.

Total running time

We analyze the total running time of our algorithm for the randomized case only since the analysis for the deterministic case is nearly identical.

The total running time for the type II transitions is $\widetilde{O}((|T_1|/\Delta)(\mathsf{MUL}(\Delta, |T_2|) + |T_2|\Delta^3))$, which equals

$$\widetilde{O}(|T_1||T_2|^2/\Delta^{0.0952} + |T_1||T_2|\Delta^2).$$

We relate the running time for the type I transitions to a value that is easier to analyze. Let X be the total running time for the type I transitions, and let Y be what the total running time for the type I transitions would be if $\mathsf{MUL}(\min(|F_1|, |F_2|, |T_2|), |T_2|)$ equaled $O(\min(|F_1|, |F_2|, |T_2|)|T_2|^2)$ instead of $\widetilde{O}(\min(|F_1|, |F_2|, |T_2|)^{0.9038}|T_2|^2)$. Since $\min(|F_1|, |F_2|, |T_2|) \geq \Delta$, $X = \widetilde{O}(Y/\Delta^{0.0952})$.

We now obtain a bound on Y. If both $|F_1|$ and $|F_2|$ are greater than $|T_2|$, the total number of such transitions is $O(|T_1|/|T_2|)$, so the total running time is $O((|T_1|/|T_2|) \times |T_2|^3) = O(|T_1||T_2|^2)$. If $\min(|F_1|, |F_2|) \leq |T_2|$, we adapt the classic argument on small to large

merging: when we are multiplying the similarity matrices for two subforests, each node in the smaller subforest contributes $O(|T_2|^2)$ to the total running time. Since the size of the smaller forest is no more than $|T_2|$, the sum of the total number of nodes in the smaller forests across all multiplications is $O(|T_1| \log|T_2|) = \widetilde{O}(|T_1|)$, and we have $Y = \widetilde{O}(|T_1||T_2|^2)$. Therefore $X = \widetilde{O}(|T_1||T_2|^2/\Delta^{0.0952})$.

The total running time for the two types combined is

$$\widetilde{O}(|T_1||T_2|^2/\Delta^{0.0952} + |T_1||T_2|\Delta^2)$$

where the hidden sub-polynomial factors are not dependent on $|T_1|$. By setting $\Delta \approx |T_2|^{0.4773}$ we get the bound in Theorem 2.1.1.

2.2.3 Transition between synchronous subforests

To get the similarity matrix for forest F given the similarity matrix for F', we need to consider mapping the nodes in $F \backslash F'$ to T_2. As shown in Figure 2-4a, we consider the path between $\mathsf{vroot}(F)$ and $\mathsf{vroot}(F')$ in forest F:

$$\mathsf{vroot}(F) = u_0 \to u_1 \to u_2 \to \cdots \to u_k = \mathsf{vroot}(F')$$

where all nodes except u_0 is in F. For $1 \le i \le k$, let l_i be the subforest consisting of subtrees of siblings of u_i to the left of u_i from left to right, and let r_i be the subforest consisting of subtrees of siblings to the right of u_i from left to right. We also let l_{k+1} be the subforest consisting of subtrees of children of u_k to the left of F', and r_{k+1} be the subforest consisting of subtrees of children of u_k to the right of F'. For simplicity, in Figure 2-4a subforests are drawn as if they were subtrees.

For $j \ge i$, let $l_{i,j}$ be the subforest $l_i + l_{i+1} + \cdots + l_j$ and let $r_{i,j}$ be the subforest $r_j + r_{j-1} + \cdots + r_i$. From Corollary 2.2.3 it is easy to see that $S(l_{i,j})$ and $S(r_{i,j})$ across all $1 \le i \le j \le k+1$ can be computed in $O(|T_2|(|F| - |F'|)^3)$ time.

If none of $u_1, u_2, \cdots, u_k$ is mapped, then the contribution to $S(F)$ will be $S(l_{1,k+1}) \star$

$S(F') \star S(r_{1,k+1})$, which can be computed in $\mathsf{MUL}(\min(|F|-|F'|,|T_2|),|T_2|)$ time. For the case where at least one of $u_1, u_2, \cdots, u_k$ is mapped, we first define a restricted version of the similarity matrix of a *tree* where the root must be mapped. For any tree T with at least one node, we let $\overline{T}$ be $T - \mathsf{root}(T)$.

Definition 2.2.6 (Restricted similarity matrix)**.** For a tree T, the *restricted similarity matrix* $\widehat{S}(T) = \widehat{s}_{i,j}(T)$ is a $(2|T_2|+1) \times (2|T_2|+1)$ matrix where for all $i \le j$ such that $T_2[i,j) \neq \emptyset$

$$\widehat{s}_{i,j}(T) = \max_{v \in T_2[i,j)} \{\mathsf{sim}(\overline{T}, \overline{\mathsf{sub}(v)}) + \eta(\mathsf{root}(T), v)\},$$

and the entries elsewhere are $-\infty$.

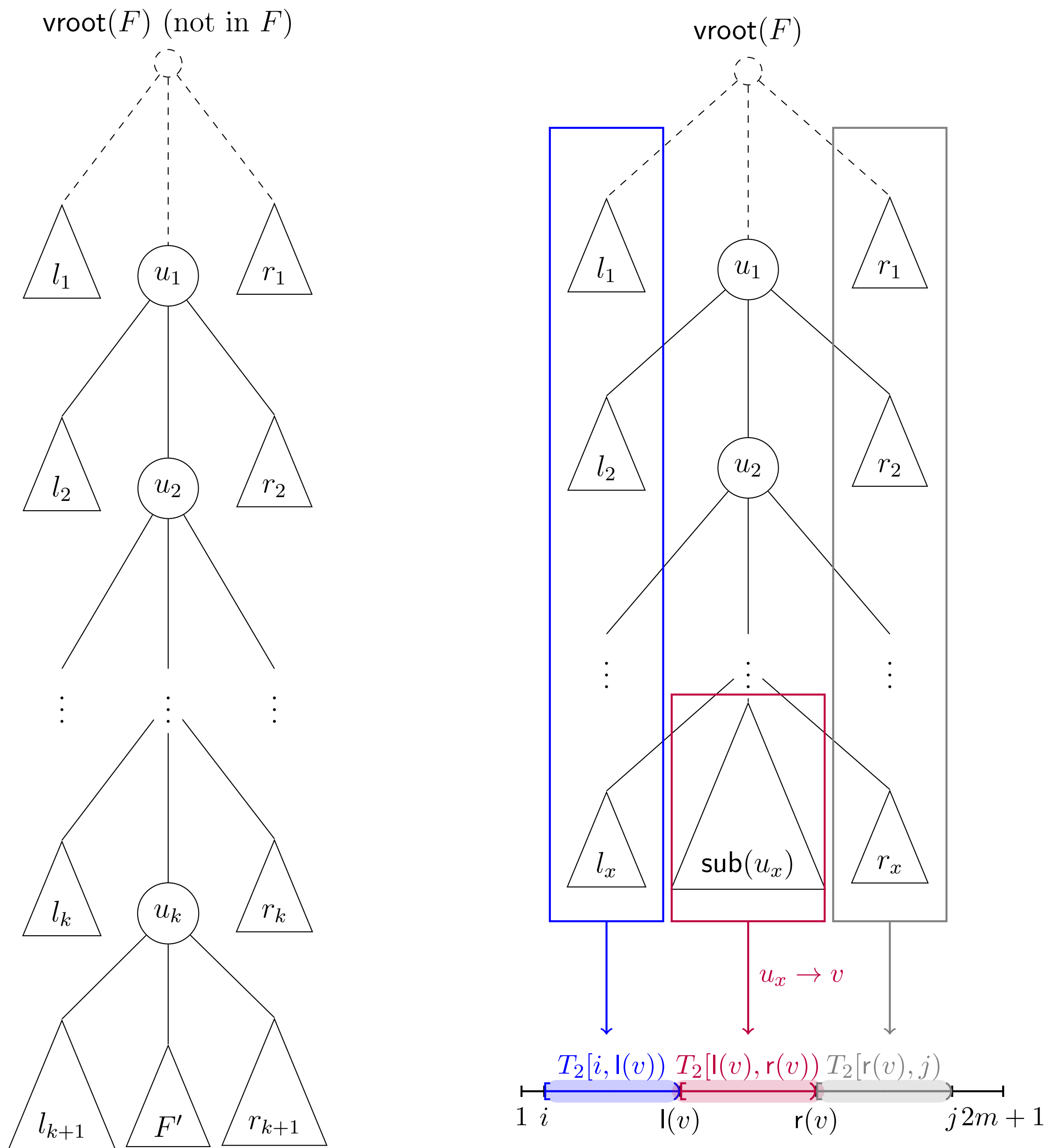
The transition from $S(F')$ to $S(F)$ consists of three parts:

- (Bottom) For the largest y such that u_y is mapped, the transition from $S(F')$ to $\widehat{S}(\mathsf{sub}(u_y))$,
- (Middle) For all (x,y) pairs where $x<y$, the transition from $\widehat{S}(\mathsf{sub}(u_y))$ to $\widehat{S}(\mathsf{sub}(u_x))$, and
- (Top) For the smallest x such that u_x is mapped, the transition from $\widehat{S}(\mathsf{sub}(u_x))$ to $S(F)$.

Final transitions from restricted similarity matrices (top)

We first show how to do the top transitions since they do not involve computation of the restricted similarity matrices. We show:

Lemma 2.2.7. *If $\widehat{S}(\mathsf{sub}(u_x))$ is known for all $1 \le x \le k$, $S(F)$ can be computed in $\widetilde{O}(\mathsf{MUL}(|F|-|F'|,|T_2|,|T_2|) + |T_2|(|F|-|F'|)^3)$ time.*



(a) The nodes $u_{1\cdots k}$ and the subforests $l_{1\cdots k+1}$ and $r_{1\cdots k+1}$

(b) Transition from $\widehat{S}(\mathsf{sub}(u_x))$ to $S(F)$ (top)

Figure 2-4: Transition between synchronous subforests

To compute $S(F)_{i,j} = \mathsf{sim}(F, T_2[i,j))$, we can enumerate the smallest x such that u_x maps to some node $v \in T_2[i,j)$. Then as shown in Figure 2-4b, we have

- Nodes in $l_{1,x}$ map to nodes in $T_2[i, \mathsf{l}(v))$, contributing $s_{i,\mathsf{l}(v)}(l_{1,x})$,
- Nodes in $\mathsf{sub}(u_x)$ map to nodes in $T_2[\mathsf{l}(v), \mathsf{r}(v))$, contributing $\widehat{s}_{\mathsf{l}(v),\mathsf{r}(v)}(\mathsf{sub}(u_x))$, and
- Nodes in $r_{1,x}$ map to nodes in $T_2[\mathsf{r}(v), j)$, contributing $s_{\mathsf{r}(v),j}(r_{1,x})$.

The total contribution will be $s_{i,\mathsf{l}(v)}(l_{1,x}) + \widehat{s}_{\mathsf{l}(v),\mathsf{r}(v)}(\mathsf{sub}(u_x)) + s_{\mathsf{r}(v),j}(r_{1,x})$. Note that $\widehat{s}_{\mathsf{l}(v),\mathsf{r}(v)}(\mathsf{sub}(u_x))$ might actually correspond to a mapping where u_x is mapped to some node other than v in $T_2[\mathsf{l}(v), \mathsf{r}(v))$, but this still gives us a valid mapping overall and does not affect the correctness of our algorithm. There are still $O(|T_2|^3(|F| - |F'|))$ quadruples (i, j, x, v), but as shown in Algorithm 3, we can cut the number of quadruples to $O(|T_2|(|F| - |F'|)^3)$ using an idea similar to the one in Algorithm 1.

Algorithm 3 Computation of $S(F)$ from $\widehat{S}(\mathsf{sub}(u_x))$

```
1:  S(F) ← S(l_{1,k+1}) ⋆ S(F') ⋆ S(r_{1,k+1})                    ▷ No u_x mapped
2:  for x ∈ [1, k] do
3:      for v ∈ T_2 do
4:          for y ∈ [0, 2(|F| − |F'|)] do
5:              for z ∈ [0, 2(|F| − |F'|)] do
6:                  i ← maxrow(S(l_{1,x}), l(v), y)
7:                  j ← mincol(S(r_{1,x}), r(v), z)
8:                  S(F) ← rangemax(S(F), i, j, s_{i,l(v)}(l_{1,x}) + ŝ_{l(v),r(v)}(sub(u_x)) + s_{r(v),j}(r_{1,x}))
9:              end for
10:         end for
11:     end for
12: end for
```

Computation of restricted similarity matrices (bottom/middle)

We now finish the proof of Theorem 2.2.5 by showing

Lemma 2.2.8. *For any $1 \le x \le k$, if $\widehat{S}(\mathsf{sub}(u_y))$ is known for all $y > x$, $\widehat{S}(\mathsf{sub}(u_x))$ can be computed in $\widetilde{O}(|T_2|(|F| - |F'|)^2)$ time.*

This enables us to compute $\widehat{S}(\mathsf{sub}(u_x))$ for all $1 \le x \le k$ in decreasing order of x with total time $O(|T_2|(|F| - |F'|)^3)$.

To compute $\widehat{S}(\mathsf{sub}(u_x))$ we need to consider two cases:

- The first case (bottom): no $u_y (y > x)$ is mapped.
- The second case (middle): there exists some $y > x$ such that u_y is mapped.

The first case (bottom) This is the simpler case. We can enumerate the triple (i, j, v) such that,

- u_x maps to v, contributing $\eta(u_x, v)$,
- Nodes in $l_{x+1,k+1}$ map to nodes in $T_2[\mathsf{l}(v) + 1, i)$, contributing $s_{\mathsf{l}(v)+1,i}(l_{x+1,k+1})$,
- Nodes in F' map to nodes in $T_2[i, j)$, contributing $s_{i,j}(F')$, and
- Nodes in $r_{x+1,k+1}$ map to nodes in $T_2[j, \mathsf{r}(v) - 1)$, contributing $s_{j,\mathsf{r}(v)-1}(r_{x+1,k+1})$.

The total contribution is $\eta(u_x, v) + s_{\mathsf{l}(v)+1,i}(l_{x+1,k+1}) + s_{i,j}(F') + s_{j,\mathsf{r}(v)-1}(r_{x+1,k+1})$, and the contribution applies to $\widehat{s}_{i',j'}(\mathsf{sub}(u_x))$ for all $i' \le \mathsf{l}(v) < \mathsf{r}(v) \le j'$.

The second case (middle) Without loss of generality we suppose no $x < y' < y$ exists such that $u_{y'}$ is mapped. We can enumerate the quadruple (i, j, y, v) such that,

- u_x maps to v, contributing $\eta(u_x, v)$,
- Nodes in $l_{x+1,y}$ map to nodes in $T_2[\mathsf{l}(v) + 1, i)$, contributing $s_{\mathsf{l}(v)+1,i}(l_{x+1,y})$,
- Nodes in $\mathsf{sub}(u_y)$ map to nodes in $T_2[i, j)$ and in particular u_y is mapped to some $v' \in T_2[i, j)$, contributing $\widehat{s}_{i,j}(\mathsf{sub}(u_y))$, and
- Nodes in $r_{x+1,y}$ map to nodes in $T_2[j, \mathsf{r}(v) - 1)$, contributing $s_{j,\mathsf{r}(v)-1}(r_{x+1,y})$.

The total contribution is $\eta(u_x, v) + s_{\mathsf{l}(v)+1,i}(l_{x+1,y}) + \widehat{s}_{i,j}(\mathsf{sub}(u_y)) + s_{j,\mathsf{r}(v)-1}(r_{x+1,y})$, and the contribution applies to $\widehat{s}_{i',j'}(\mathsf{sub}(u_x))$ for all $i' \leq \mathsf{l}(v) < \mathsf{r}(v) \leq j'$.

Using an idea similar to the one in Algorithm 1, the number of triples in the first case can be reduced to $O(|T_2|(|F| - |F'|)^2)$, while the number of quadruples in the second case can be reduced to $O(|T_2|(|F| - |F'|)^3)$, which is just an extra $O(|F| - |F'|)$ factor more than the bound in Lemma 2.2.8. In fact, this extra $O(|F| - |F'|)$ overhead in quadruple count will only change the $|T_2|(|F| - |F'|)^3$ term in Theorem 2.2.5 to $|T_2|(|F| - |F'|)^4$, and one can still achieve a truly sub-cubic running time overall by setting the block size $\Delta = O(|T_1|^c)$ for any $0 < c < \frac{1}{3}$.

We note that on T_2, v must be an ancestor of the node v' that u_y maps to. It can be shown that by combining this fact with some application with a data structure that supports path modifications on a tree (i.e. *link/cut tree* [43]), we can speed up the running time for the second case to $\widetilde{O}(|T_2|(|F| - |F'|)^2)$.

Chapter 3

Faster Approximation Schemes for Partition and Knapsack

3.1 Introduction

3.1.1 Background

Knapsack, *Subset Sum*, and *Partition* are three fundamental problems in computer science and mathematical optimization, and are actively being studied in fields such as integer programming and fine-grained complexity. In the Knapsack problem (sometimes also called 0-1 Knapsack), we are given a set I of n items where each item $i \in I$ has weight $w_i > 0$ and profit $p_i > 0$, as well as a knapsack capacity W, and we want to choose a subset $J \subseteq I$ satisfying the weight constraint $\sum_{j \in J} w_j \leq W$ such that the total profit $\sum_{j \in J} p_j$ is maximized. The Subset Sum problem is a special case of Knapsack, where the weight of an item is always equal to their profit. The Partition problem is a special case of Subset Sum, where the capacity equals half of the total weight of the items. In other words, in Partition we want to partition the input items into two parts so that their sums is as close as possible.

These three problems are well-known to be hard: they appeared in Karp's original list of 21 NP-hard problems [27]. To cope with NP-hardness, a natural direction is to study their *approximation algorithms*. Given a parameter $\varepsilon > 0$, and an input instance with optimal

value OPT, a $(1-\varepsilon)$-approximation algorithm is required to output a number SOL such that $(1-\varepsilon)\text{OPT} \le \text{SOL} \le \text{OPT}$. Fortunately, these three problems are well-known to have *fully polynomial-time approximation schemes (FPTASes)*, namely $(1-\varepsilon)$-approximation algorithm that runs in $\text{poly}(n, 1/\varepsilon)$ time, for any $\varepsilon > 0$.

There has been a long line of research since the 70's on getting approximation schemes for these problems with improved time complexities in terms of n and $1/\varepsilon$ [24, 35, 20, 29, 28, 30, 38, 25, 11, 37, 26, 8]. On the other hand, recent advances in fine-grained complexity have pointed out the limit of such improvements, under well-believed hardness assumptions [16, 34, 1, 8]. Here, we briefly describe the most recent results along this line.

- **Knapsack:** The best known algorithm by Jin [26] runs in $\widetilde{O}(n + \varepsilon^{-2.25})$ time, and is based on the previous algorithm of Chan [11] in $\widetilde{O}(n + \varepsilon^{-2.4})$ time. [16] and [34] showed a conditional lower bound of $(n + \frac{1}{\varepsilon})^{2-o(1)}$, based on the $(\min, +)$-convolution hypothesis.

- **Subset Sum:** The best known algorithm by Bringmann and Nakos [8] runs in $\widetilde{O}(n + \varepsilon^{-2}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ time (improving [28] by low-order factors). Bringmann and Nakos [8] showed a matching lower bound based on the $(\min, +)$-convolution hypothesis.

- **Partition:** The first breakthrough by Mucha, Węgrzycki and Włodarczyk [37] gave a randomized algorithm in $\widetilde{O}(n+1/\varepsilon^{5/3})$ time. Later, Bringmann and Nakos [8] improved it to deterministic $\widetilde{O}(n + 1/\varepsilon^{-3/2})$ time. Abboud, Bringmann, Hermelin, and Shabtay [1] showed that Partition cannot be approximated in $\text{poly}(n)/\varepsilon^{1-\delta}$ time for any $\delta > 0$, under the Strong Exponential Time Hypothesis.

We can see that the complexity of Subset Sum is already settled, but for Knapsack and Partition there still remain gaps between the best known algorithms and their conditional lower bounds.

3.1.2 Our results

In this chapter, we show two recent progresses on this direction, by giving improved approximation schemes for Partition and Knapsack.

Theorem 3.1.1. *There is a deterministic algorithm for* $(1-\varepsilon)$*-approximating Partition with running time*

$$\widetilde{O}\left(n+\varepsilon^{-5/4}\right).$$

Theorem 3.1.2. *There is a randomized algorithm for* $(1-\varepsilon)$*-approximating Knapsack with running time*

$$\widetilde{O}\left(n+\varepsilon^{-11/5}/2^{\Omega(\sqrt{\log(1/\varepsilon)})}\right).$$

3.2 Preliminaries

We write $\mathbb{N}=\{0,1,2,\dots\}$ and $\mathbb{N}^+=\{1,2,\dots\}$. For $n\in\mathbb{N}$ we write $[n]=\{1,2,\dots,n\}$.

3.2.1 Problem Statements

In the Knapsack problem, the input is a list of n items $(p_1,w_1),\dots,(p_n,w_n)\in\mathbb{N}\times\mathbb{N}$ together with a knapsack capacity $W\in\mathbb{N}$, and the optimal value is

$$\mathrm{OPT}:=\max_{J\subseteq[n]}\Big\{\sum_{j\in J}p_j\,\Big|\,\sum_{j\in J}w_j\le W\Big\}.$$

In the easier Partition problem, the input is a list of n integers $x_1,\dots,x_n\in\mathbb{N}$, and the optimal value is

$$\mathrm{OPT}:=\max_{J\subseteq[n]}\Big\{\sum_{j\in J}x_j\,\Big|\,\sum_{j\in J}x_j\le\frac{1}{2}\sum_{i\in[n]}x_i\Big\}.$$

Given a Knapsack (or a Partition) instance and a parameter $\varepsilon\in(0,1)$, an $(1-\varepsilon)$-*approximation algorithm* is required to output a number SOL such that $(1-\varepsilon)\mathrm{OPT}\le\mathrm{SOL}\le\mathrm{OPT}$.

In both problems, we can assume $n = O(\varepsilon^{-4})$ and hence $\log n = O(\log \varepsilon^{-1})$. For larger n, Lawler's algorithm [35] for Knapsack in $O(n \log \frac{1}{\varepsilon} + (\frac{1}{\varepsilon})^4)$ time is already near-optimal.

We will sometimes describe algorithms with approximation ratio $1 - O(\varepsilon)$ (or $1 - \varepsilon \cdot \operatorname{poly}\log(1/\varepsilon)$), which can be made $1 - \varepsilon$ by scaling down ε by a constant factor (or a logarithmic factor) at the beginning.

3.2.2 Sumsets and Subset Sums

In a multiset A, an element a could appear multiple times (the number of times it appears is the *multiplicity* of a in A). We use $A \uplus B$ to denote union without removing duplicates (i.e., possibly resulting in a multiset).

For a multiset $Y \subset \mathbb{N}$, let $\Sigma(Y) = \sum_{y \in Y} y$ denote the sum of its elements (without removing duplicates).

For a multiset $X \subset \mathbb{N}$, let $\mathcal{S}(X) = \{\Sigma(Y) : Y \subseteq X\}$ be the set of its subset sums, and let $\mathcal{S}(X; t) = \mathcal{S}(X) \cap [0, t]$ be the set of its subset sums up to t.

For a number c and a set X, define $c \cdot X = \{cx : x \in X\}$. For two sets X, Y, define their *sumset* $X + Y = \{x + y : x \in X, y \in Y\}$. Given sets $X \subseteq [n], Y \subseteq [n]$, the sumset $X + Y$ can be computed in $O(n \log n)$ time using FFT. This simple fact has a straightforward generalization to 2 dimension, which we state below.

Lemma 3.2.1 (2-dimensional FFT, e.g., [4, Chapter 12.8])**.** *Given two sets $A_1, A_2 \subseteq [n] \times [m]$, one can compute*

$$A_1 + A_2 := \big\{(x_1 + x_2, y_1 + y_2) : (x_1, y_1) \in A_1, (x_2, y_2) \in A_2\big\}$$

in $O(nm \log(nm))$ time deterministically.

3.2.3 Knapsack Problem and Profit functions

In the knapsack problem, assume $0 < w_i \le W$ and $p_i > 0$ for every item i. Then a trivial lower bound of the maximum total profit is $\max_j p_j$. At the beginning, we can discard all

items i with $p_i \le \frac{\varepsilon}{n} \max_j p_j$, reducing the total profit by at most $\varepsilon \max_j p_j$, which is only an $O(\varepsilon)$ fraction of the optimal total profit. So we can assume $\frac{\max_j p_j}{\min_j p_j} \le \frac{n}{\varepsilon}$.

For a set I of items, we use f_I to denote its *profit function*, defined as

$$f_I(x) = \max \left\{ \sum_{i \in J} p_i : \sum_{i \in J} w_i \le x, \quad J \subseteq I \right\}$$

over $x \in [0, +\infty)$. Note that f_I is a monotone nondecreasing step function. Adopting the terminology of Chan [11], the *complexity* of a monotone step function refers to the number of its steps.

Let I_1, I_2 be two disjoint subsets of items, and $I = I_1 \uplus I_2$. It is straightforward to see that $f_I = f_{I_1} \oplus f_{I_2}$, where $\oplus$ denotes $(\max, +)$-convolution, defined by $(f \oplus g)(x) = \max_{0 \le x' \le x}(f(x') + g(x - x'))$.

3.2.4 $(1 - \delta, \Delta)$ approximation up to t

Both our algorithms for Knapsack and Partition frequently use the notion of $(1 - \delta, \Delta)$-*approximation up to t*. Their definitions are analogous, as stated below.

Definition 3.2.2 (Approximation for Profit Functions)**.** For functions $\widetilde{f}, f$ and real numbers $t, \Delta \in \mathbb{R}_{\ge 0}, \delta \in [0, 1)$, we say that $\widetilde{f}$ is a $(1 - \delta, \Delta)$ *approximation of f up to t*, if

$$\widetilde{f}(w) \le f(w)$$

holds for all $w \ge 0$, and

$$\widetilde{f}(w) \ge (1 - \delta) f(w) - \Delta$$

holds whenever $f(w) \le t, w \ge 0$.

The following notion of approximation will be useful in our Partition algorithm. Similar notions have been termed as "weak approximation" in the literature [37, 8], in contrast to "strong approximation" that would be required for approximating general Subset Sum instances.

Definition 3.2.3 (Approximation for Integer Sets)**.** For integer sets $A, B \subseteq \mathbb{N}$, and real numbers $t, \Delta \in \mathbb{R}_{\geq 0}, \delta \in [0, 1)$, we say that A is a $(1-\delta, \Delta)$ *approximation of B up to t*, if

1. for every $b \in B \cap [0, t]$, there exists $a \in A$ such that $(1-\delta)b - \Delta \leq a \leq b$, and,
2. for every $a \in A$, there exists $b \in B$ such that $(1-\delta)b - \Delta \leq a \leq b$.

One can assume $A \subseteq \mathbb{N} \cap [0, t]$ in this case without loss of generality.

For the case of $t = +\infty$, we simply omit the phrase "up to t".

We also refer to $(1, \Delta)$ approximation as Δ-*additive approximation*, and refer to $(1-\delta, 0)$ approximation as $(1-\delta)$-*multiplicative approximation*, or simply $(1-\delta)$ *approximation*.

We have the following simple facts regarding approximating merged sumsets and profit functions.

Proposition 3.2.4. *For $i \in \{1, 2\}$, suppose A_i is a $(1-\delta, \Delta_i)$ approximation of $\mathcal{S}(X_i)$ up to t. Then, $(A_1 + A_2) \cap [0, t]$ is a $(1-\delta, \Delta_1 + \Delta_2)$ approximation of $\mathcal{S}(X_1 \uplus X_2)$ up to t.*

Proof. For any $b \in \mathcal{S}(X_1 \uplus X_2) \cap [0, t]$ where $b = b_1 + b_2$ for $b_i \in \mathcal{S}(X_i) \cap [0, t]$ $(i \in \{1, 2\})$, there exits $a_i \in A_i$ such that $(1-\delta)b_i - \Delta_i \leq a_i \leq b_i$. Hence, $a_1 + a_2 \leq b_1 + b_2 = b \leq t$, and $a_1 + a_2 \geq (1-\delta)b_1 - \Delta_1 + (1-\delta)b_2 - \Delta_2 = (1-\delta)b - (\Delta_1 + \Delta_2)$.

The converse direction can be verified similarly. □

The following fact can be proved similarly.

Proposition 3.2.5. *For $i \in \{1, 2\}$, suppose $\widetilde{f_i}$ is a $(1-\delta, \Delta_i)$ approximation of the profit function f_{I_i} up to t. Then, $(\widetilde{f_1} \oplus \widetilde{f_2})$ is a $(1-\delta, \Delta_1 + \Delta_2)$ approximation of $f_{I_1 \uplus I_2}$ up to t.*

Following Chan [11] and Jin [26], given a monotone step function f (we sometimes also call it a profit function, although it might not be equal to the profit function f_I of any particular item set I) with range contained in $\{0\} \cup [A, B]$, one can round f down to powers of $1/(1-\varepsilon)$, and obtain another profit function $\widetilde{f}$ which has complexity only $O(\varepsilon^{-1} \log(B/A))$, and $(1-\varepsilon)$-approximates f. In our algorithm we will always have $B/A \leq \mathrm{poly}(n/\varepsilon)$, so we may always assume that the intermediate profit functions computed during our algorithm

are monotone step functions with complexity $\widetilde{O}(\varepsilon^{-1})$, by incurring $(1-\varepsilon)$ approximation factor each time.

3.2.5 Implicit Dynamic Progamming Schemes

In spirit of the theme of this thesis, we demonstrate the roles dynamic programming will play in our algorithms. In the classic dynamic programming solution for Subset Sum with the input set $\{A_1, A_2, \cdots, A_n\}$, we store a Boolean table $a_{i,j}$ which is true if and only if the sum j can be achieved with the first i elements in the set. For a fixed i, the set of indices j where $a_{i,j}$ is true is the set $\mathcal{S}(\{A_1, A_2, \cdots, A_i\})$. For the knapsack problem, the classic dynamic programming stores the values $v_{i,j}$ which is the maximum value one can achieve for a subset of the first i elements with weights summing to at most j. For a fixed i, this corresponds to the profit function $f_i(x) = v_{i,x}$. By performing operations on sumsets and profit functions, we are in fact doing dynamic programming implicitly, and these implicit dynamic programming schemes will play an important part at the core of our algorithms.

3.2.6 Additive Combinatorics

We need several results on dense subset sums developed by a series of works including [39, 36, 19, 9]. The following structural lemma follows from Theorem 4.1 and Theorem 4.2 of Bringmann and Wellnitz [9].

Lemma 3.2.6. *Let n distinct positive integers $X = \{x_1, \ldots, x_n\} \subseteq [\ell, 2\ell]$ be given, where $\ell = o(n^2/\log n)$.*

Then, for a universal constant $c \geq 1$, for every $c\ell^2/n \leq t \leq \Sigma(X)/2$, there exists $t' \in \mathcal{S}(X)$ such that $0 \leq t' - t \leq 8\ell/n$.

A proof of Lemma 3.2.6 is included in the full paper.

The following algorithmic lemma follows from the main theorem of [9].

Lemma 3.2.7 (Follows from [9]). *Given n distinct positive integers $X = \{x_1, \ldots, x_n\} \subseteq [\ell, 2\ell]$, there exists $\lambda = \widetilde{\Theta}(\ell^2/n)$ such that, if $\lambda \leq \Sigma(X)/2$, then in $\widetilde{O}(n)$ time we can*

construct a deterministic data structure supporting the following query in $O(1)$ time: given L, R such that $\lambda \le L \le R \le \Sigma(X)/2$, report whether there exists $t \in [L, R]$ such that $t \in \mathcal{S}(X)$.

Remark 3.2.8. We remark that the main theorem stated in [9] only supports querying whether a given $\lambda \le t \le \Sigma(X)/2$ is a subset sum. In our application, we require a version supporting range queries. This is easy to achieve by building an additional prefix sum array in the proof of [9, Theorem 4.6], which supports range sum queries.

3.3 Approximating Partition

In this section, we will solve the following problem.

Problem 1. *Assume $\varepsilon \in (0, 1/2)$ and $1/\varepsilon \in \mathbb{N}^+$. Given a set X of n* distinct *integers in the interval $[1/\varepsilon, 2/\varepsilon)$, compute a set $A \subset \mathbb{N}$ that n-additively approximates $\mathcal{S}(X)$.*

By a tedious reduction that is heavily based on known techniques, one can show the following.

Lemma 3.3.1. *If for some $c \ge 1$, Problem 1 can be solved in $\widetilde{O}(n + 1/\varepsilon^c)$ time, then $(1-\varepsilon)$-approximating* Partition *can also be solved in $\widetilde{O}(n + 1/\varepsilon^c)$ time.*

Now we proceed to describe our main algorithm for solving Problem 1.

In the following lemma, we merge the approximations of $\mathcal{S}(X_1), \mathcal{S}(X_2)$ and obtain an approximation of $\mathcal{S}(X_1 \uplus X_2)$. When X_1, X_2 come from a short interval $[\ell, \ell + d]$, we can use densification via 2D FFT to obtain a speed-up over the straightforward algorithm.

Lemma 3.3.2. *Let $\delta \in (0, 1/2)$, and $\ell, d, t, \Delta \in \mathbb{N}^+$ such that $d \le \ell \le t$.*

Let $X_1, X_2 \subseteq \mathbb{N}^+ \cap [\ell, \ell + d]$ be two integer sets. Given $A_1, A_2 \subset \mathbb{N}$ as input where for $i \in \{1, 2\}$, A_i is an $(1-\delta)$ approximation of $\mathcal{S}(X_i)$ up to t, one can compute a set $A \subset \mathbb{N}^+$ of size $|A| \le Z$ that $(1-\delta, \Delta - 1)$-approximates $\mathcal{S}(X_1 \uplus X_2)$ up to t, in $\widetilde{O}(Z + |A_1| + |A_2|)$ time, where

$$Z \le O\left(\min\left\{\left\lceil \frac{t}{\Delta} \right\rceil, \ \frac{t}{\ell} \cdot \left\lceil \frac{td}{\ell\Delta} \right\rceil \right\}\right).$$

Proof. Let $\bar{\Delta} := \lceil \Delta/2 \rceil$. We will run one of the following two algorithms that minimizes Z.

Algorithm 1 (1D FFT). For $i \in \{1,2\}$, by rounding the integers in A_i down to multiples of $\bar{\Delta}$, we obtain set $A'_i \subset \bar{\Delta} \cdot \mathbb{N}$ that $(\bar{\Delta}-1)$-additively approximates A_i. Then, since $A'_i \subseteq [0,t]$, their sumset $A'_1 + A'_2$ can be computed by FFT in $\widetilde{O}(\lceil t/\bar{\Delta} \rceil) \le \widetilde{O}(\lceil t/\Delta \rceil)$ time. Note that $A := A'_1 + A'_2$ approximates $A_1 + A_2$ with additive error at most $2(\bar{\Delta}-1) \le \Delta - 1$, so A is a $(1-\delta, \Delta-1)$-approximation of $\mathcal{S}(X_1 \uplus X_2)$ up to t, due to Proposition 3.2.4.

Algorithm 2 (Densification with 2D FFT). For every $a \in A_i$, there exists $s \in \mathcal{S}(X_i; t)$ such that $0 \le s - a \le s\delta$. Note that s is the sum of at most t/ℓ many integers from $[\ell, \ell + d]$, so s can be expressed as $s = k\ell + b'$ for some $k \in \mathbb{N} \cap [0, t/\ell]$ and $0 \le b' \le dt/\ell$. Hence, $a \in A_i$ can be expressed as $a = k\ell + b$ for some $k \in \mathbb{N} \cap [0, t/\ell]$ and $-s\delta \le b \le dt/\ell$. Then, by rounding b down to integer multiples of $\bar{\Delta}$, we obtain $A'_i \subset \mathbb{N}$ that $(\bar{\Delta}-1)$-additively approximates A_i, such that every $a' \in A'_i$ can be expressed as

$$a' = k\ell + j\bar{\Delta},$$

for some $k \in \mathbb{N} \cap [0, t/\ell]$ and $j \in \mathbb{Z} \cap [-1 - s\delta/\bar{\Delta}, dt/(\ell\bar{\Delta})]$. Using this 2-dimensional (k, j) representation of A'_i, we can compute $A'_1 + A'_2$ using 2D FFT (Lemma 3.2.1): the first dimension has size $O(t/\ell)$, and the second dimension has size at most

$$dt/(\ell\bar{\Delta}) + s\delta/\bar{\Delta} + O(1) \le O\left(\left\lceil \frac{td}{\ell\Delta} \right\rceil\right),$$

where the inequality follows from $s \le t$ and an assumption

$$\delta \le O(d/\ell), \tag{3.1}$$

which will be justified later. Hence, the running time of this 2D FFT is

$$\widetilde{O}\left(\frac{t}{\ell} \cdot \left\lceil \frac{td}{\ell\Delta} \right\rceil\right).$$

Similarly to Algorithm 1, one also can show that in this case $A := A'_1 + A'_2$ is a $(1-\delta, \Delta-1)$-approximation of $\mathcal{S}(X_1 \uplus X_2)$ up to t.

To justify assumption (3.1), observe that if $\delta \geq d/(\ell+d)$ holds instead, or equivalently, $(1-\delta)(\ell+d) \leq \ell$, then one can round every integer in $X_1, X_2 \subset [\ell, \ell+d]$ down to exactly ℓ while still ensuring $(1-\delta)$ approximation, and hence immediately obtain an $A \subset \ell \cdot \mathbb{N}$ of size $|A| \leq \lceil t/\ell \rceil$ that $(1-\delta)$-approximates $\mathcal{S}(X_1 \uplus X_2)$ up to t. □

We then apply Lemma 3.3.2 with scaling, and obtain the following lemma that has purely multiplicative approximation.

Lemma 3.3.3. *Let $\delta, \delta_0 \in (0, 1/2)$, and $\ell, d, T \in \mathbb{N}^+$ such that $d \leq \ell \leq T$.*

Let $X_1, X_2 \subseteq \mathbb{N}^+ \cap [\ell, \ell+d]$ be two integer sets. Given $A_1, A_2 \subset \mathbb{N}$ as input where for $i \in \{1, 2\}$, A_i is an $(1-\delta)$ approximation of $\mathcal{S}(X_i)$ up to T, one can compute a set $A \subset \mathbb{N}^+$ of size $|A| \leq Z$ that $(1-\delta-\delta_0)$-approximates $\mathcal{S}(X_1 \uplus X_2)$ up to T, in $\widetilde{O}\big(Z + (|A_1|+|A_2|)\log(2T/\ell)\big)$ time, where

$$Z \leq O\left(\min\left\{\frac{\log(2T/\ell)}{\delta_0},\ \frac{T}{\ell} \cdot \left\lceil \frac{d}{\ell\delta_0} \right\rceil\right\}\right).$$

Proof. Initialize set $A = \{0\}$. We iterate over all r being integer powers of 2 such that $\ell/6 \leq r \leq T$. For each r, apply Lemma 3.3.2 to A_1 and A_2 with $t := 6r$ and $\Delta := \lceil \delta_0 r \rceil$, and obtain a set $A_r \subseteq \mathbb{N} \cap [0, 6r]$ that $(1-\delta, \lceil \delta_0 r \rceil - 1)$-approximates $\mathcal{S}(X_1 \uplus X_2)$ up to $6r$. We then insert all elements in $A_r \cap [r, 6r]$ into A. We will show that eventually A is a $(1-\delta_0-\delta)$-approximation of $\mathcal{S}(X_1 \uplus X_2)$ up to T.

Observe that for every $a \in A_r \cap [r, 6r]$, there exists $s \in \mathcal{S}(X_1 \uplus X_2)$ such that $a \leq s$ and

$$\begin{aligned} a &\geq (1-\delta)s - (\lceil \delta_0 r \rceil - 1) \\ &> (1-\delta)s - \delta_0 r \\ &\geq (1-\delta-\delta_0)s, \end{aligned}$$

where the last step follows from $s \geq a \geq r$.

Conversely, for every positive $s \in \mathcal{S}(X_1 \uplus X_2; T)$ (which must satisfy $\ell \leq s \leq T$), let r be

a power of two such that $3r \le s \le 6r$. Then there exists $a \in A_r$ such that $a \le s \le 6r$ and

$$\begin{aligned} a &\ge (1-\delta)s - (\lceil \delta_0 r \rceil - 1) \\ &\ge (1-\delta)s - \delta_0 r \\ &\ge s/2 - r/2 \\ &\ge r, \end{aligned}$$

so $a \in A_r \cap [r, 6r]$ and hence will be included in A, and similarly as before we have $a \ge (1-\delta_0-\delta)s$. Hence, we have established that A is a $(1-\delta_0-\delta)$-approximation of $\mathcal{S}(X_1 \uplus X_2)$ up to T.

It remains to bound the total running time and the size of A. There are $O(\log(2T/L))$ many iterations of r, where for each $r \in [\ell/6, T]$ with $t := 6r$ and $\Delta := \lceil \delta_0 r \rceil$, Lemma 3.3.2 gives the upper bound

$$\begin{aligned} Z_r &\le O\left(\min\left\{\left\lceil \frac{t}{\Delta} \right\rceil, \frac{t}{\ell} \cdot \left\lceil \frac{td}{\ell\Delta} \right\rceil\right\}\right) \\ &\le O\left(\min\left\{\left\lceil \frac{r}{\delta_0 r} \right\rceil, \frac{r}{\ell} \cdot \left\lceil \frac{rd}{\ell\delta_0 r} \right\rceil\right\}\right) \\ &\le O\left(\min\left\{\left\lceil \frac{1}{\delta_0} \right\rceil, \frac{r}{\ell} \cdot \left\lceil \frac{d}{\ell\delta_0} \right\rceil\right\}\right). \end{aligned}$$

Hence, summing over all powers of two in the range $[\ell/6, T]$, we have

$$Z \le \sum_r Z_r \le O\left(\min\left\{\frac{\log(2T/\ell)}{\delta_0}, \frac{T}{\ell} \cdot \left\lceil \frac{d}{\ell\delta_0} \right\rceil\right\}\right). \qquad \square$$

Lemma 3.3.3 implies the following immediate corollary by dropping the upper bound T.

Corollary 3.3.4. *Let $\delta, \delta_0 \in (0, 1/2)$, and $\ell, d \in \mathbb{N}^+$ such that $d \le \ell$.*

Let $X_1, X_2 \subseteq \mathbb{N}^+ \cap [\ell, \ell + d]$ be two integer sets of total size $|X_1| + |X_2| = n$. Given $A_1, A_2 \subset \mathbb{N}$ as input where for $i \in \{1, 2\}$, A_i is an $(1-\delta)$ approximation of $\mathcal{S}(X_i)$, one can compute a set $A \subset \mathbb{N}^+$ of size $|A| \le Z$ that $(1-\delta_0-\delta)$-approximates $\mathcal{S}(X_1 \uplus X_2)$, in

$\widetilde{O}\big(Z + (|A_1| + |A_2|)\log n\big)$ *time, where*

$$Z \le O\left(\min\left\{\frac{1}{\delta_0},\ \frac{nd}{\ell\delta_0} + n\right\} \cdot \log n\right).$$

Proof. Immediately follows from Lemma 3.3.3 by setting $T = n \cdot (\ell + d)$, which is an upper bound on the largest element of $\mathcal{S}(X_1 \uplus X_2)$. □

Now, we apply Corollary 3.3.4 in a divide-and-conquer fashion, to approximate the subset sums of $X \subseteq \mathbb{N}^+ \cap [\ell, 2\ell]$.

Lemma 3.3.5. *Let $\delta \in (0, 1/2)$ and $\ell \in \mathbb{N}^+$.*

Given an integer set $X \subseteq \mathbb{N}^+ \cap [\ell, 2\ell]$ of n integers, one can compute a set $A \subset \mathbb{N}^+$ that $(1-\delta)$-approximates $\mathcal{S}(X)$, in $\widetilde{O}(n + \sqrt{n}/\delta)$ time.

Proof. Let $X = \{x_1, x_2, \ldots, x_n\}$ where $\ell \le x_1 < x_2 < \cdots < x_n \le 2\ell$. Set $\delta_0 := \delta/\lceil \log_2 n \rceil$.

We will use a divide-and-conquer approach to merge the items of X using Corollary 3.3.4. Build a balanced binary tree with n leaf nodes representing the items $x_1, \ldots, x_n$ from left to right. At each internal node representing $x_{[l..r]}$, we use Corollary 3.3.4 to merge the results of the two child nodes (representing $x_{[l..m]}$ and $x_{[m+1..r]}$ respectively, where $m = \lfloor (l+r)/2 \rfloor$), and obtain an approximation of $\mathcal{S}(\{x_l, x_{l+1}, \ldots, x_r\})$. Finally we obtain an approximation of $\mathcal{S}(X)$ at the root node.

The binary tree has $\lceil \log_2 n \rceil$ levels, where each level of applying Corollary 3.3.4 worsens the approximation factor by δ_0. Hence, the overall approximation factor of $\mathcal{S}(X)$ is $1 - \delta_0 \cdot \lceil \log_2 n \rceil = 1 - \delta$ as required.

It remains to bound the total running time of all invocations of Corollary 3.3.4. Note that in each invocation, the $(|A_1| + |A_2|)\log n$ summand in the stated time complexity is always absorbed (up to $\log n$ factors) by the output sizes of the two child nodes, which are in turn bounded by the running times of these two child invocations. So it suffices to bound the sum of the Z quantity stated in Corollary 3.3.4.

We separately bound for each level of the binary tree. At the i-th level ($0 \le i < \lceil \log_2 n \rceil$), there are at most $m_i = 2^i$ invocations of Corollary 3.3.4, where each invocation involves at

most $n_i = \lceil n/2^i \rceil$ items in X. Note that $n_i m_i \le 2n$. Suppose these m_i invocations involve $x_{[1..k_1]}, x_{[k_1+1..k_2]}, \dots, x_{[k_{m_i-1}+1..n]}$ respectively. Then the j-th invocation has d value (stated in Corollary 3.3.4) at most $d_j \le x_{k_j} - x_{k_{j-1}}$. Hence, the sum of these d values is at most

$$\sum_{j=1}^{m_i} d_j \le \sum_{j=1}^{m_i} (x_{k_j} - x_{k_{j-1}}) \le x_n - x_1 \le \ell. \tag{3.2}$$

Now we are ready to bound the sum of the Z quantity over the m_i invocations at level i ($0 \le i < \lceil \log_2 n \rceil$). We consider two cases.

- **Case 1**: $n_i \le \sqrt{n}$.

  Then, by Corollary 3.3.4,

$$\begin{aligned}
\sum_{j=1}^{m_i} Z_j &\le \sum_{j=1}^{m_i} \left(\frac{n_i d_j}{\ell \delta_0} + n_i \right) \cdot \log n \\
&= \left(\frac{n_i \sum_{j=1}^{m_i} d_j}{\ell \delta_0} + m_i n_i \right) \cdot \log n \\
&\le \left(\frac{n_i}{\delta_0} + m_i n_i \right) \cdot \log n && \text{(by (3.2))} \\
&\le \widetilde{O}\left(\frac{\sqrt{n}}{\delta_0} + n \right).
\end{aligned}$$

- **Case 2:** $n_i > \sqrt{n}$.

  Then, $m_i \le 2n/n_i < 2\sqrt{n}$. By Corollary 3.3.4,

$$\begin{aligned}
\sum_{j=1}^{m_i} Z_j &\le m_i \cdot \frac{1}{\delta_0} \cdot \log n \\
&\le \widetilde{O}(\sqrt{n}/\delta_0).
\end{aligned}$$

Hence, in either case we have $\sum_{j=1}^{m_i} Z_j \le \widetilde{O}(n + \sqrt{n}/\delta)$. Hence, the total running time over all levels $0 \le i < \lceil \log_2 n \rceil$ is also $\widetilde{O}(n + \sqrt{n}/\delta)$. □

Finally, we solve Problem 1 by combining Lemma 3.3.5 with the additive combinatorics

results of [19, 9].

Lemma 3.3.6. *We can solve Problem 1 in* $\widetilde{O}\left(n + \min\{\varepsilon^{-1}n^{1/2},\ \varepsilon^{-1} + \varepsilon^{-2}/n^{3/2}\}\right)$ *time, which is at most* $\widetilde{O}(n + 1/\varepsilon^{5/4})$ *.*

Proof. Recall that in Problem 1, for $\varepsilon > 0$ where $\ell = 1/\varepsilon$ is an integer, we are given a set $X \subseteq \mathbb{N}^+ \cap [\ell, 2\ell)$ of n distinct integers, and need to compute a set $A \subset \mathbb{N}$ that n-additively approximates $\mathcal{S}(X)$.

We will choose to run one of the following two algorithms depending on the parameters.

Algorithm 1. Directly apply Lemma 3.3.5 with $\delta := \varepsilon$, in $\widetilde{O}(n + \sqrt{n}/\varepsilon)$ time.

When $n \leq \widetilde{O}(1/\varepsilon^{1/2})$, the running time of Algorithm 1 is $\widetilde{O}(n + 1/\varepsilon^{5/4})$.

Algorithm 2. Let $\sigma = \Sigma(X)$, and let λ be the threshold value from Theorem 3.2.7 satisfying $\lambda = \widetilde{\Theta}(\ell^2/n)$. The following algorithm applies when $\lambda \leq \sigma/2$, which holds in particular when $1/\varepsilon \ll n^2$.

Initialize $A = \emptyset$. We set $\delta := n/(n + \lambda)$, and apply Lemma 3.3.5 in $\widetilde{O}(n + \sqrt{n}/\delta)$ time to compute a set A_δ that $(1-\delta)$-approximates $\mathcal{S}(X)$. Observe that $A_\delta \cap [0, \lambda]$ is an n-additive approximation of $\mathcal{S}(X)$ up to λ. Hence, we insert all elements in $A_\delta \cap [0, \lambda]$ to A.

Then, using the data structure from Lemma 3.2.7, we compute an n-additive approximation of $\mathcal{S}(X) \cap [\lambda, \sigma/2]$ and insert them into A. To do this, we start from the left endpoint λ of the interval $[\lambda, \sigma/2]$, and each time use binary search (implementable using range queries supported by Lemma 3.2.7) to find the next subset sum in the interval, and then jump n steps to the right since we allow an additive error of n. The time complexity is $O(\lceil \frac{\sigma/2-\lambda}{n} \rceil \cdot \log \sigma) \leq \widetilde{O}(\ell)$.

Now we have constructed A that n-additively approximates $\mathcal{S}(X)$ up to $\sigma/2$. Using the simple fact that $t \in \Sigma(X)$ if and only if $\sigma - t \in \Sigma(X)$, we can symmetrically use A to obtain an approximation of the remaining half. Specifically, letting $A' := \{\sigma - a - n : a \in A\}$, it is straightforward to verify that $A \cup A'$ is an n-additive approximation of $\mathcal{S}(X)$ (up to σ). So we return $A \cup A'$.

The overall time complexity of Algorithm 2 is

$$\widetilde{O}(n + \ell + \sqrt{n}/\delta) \leq \widetilde{O}\left(n + 1/\varepsilon + \frac{\sqrt{n}(n+\lambda)}{n}\right)$$
$$\leq \widetilde{O}\left(n + 1/\varepsilon + \frac{1/\varepsilon^2}{n^{3/2}}\right).$$

When $n \gg 1/\varepsilon^{1/2}$, the running time is $\widetilde{O}(n + 1/\varepsilon^{5/4})$. □

Combined with the reduction in Lemma 3.3.1, this proves our main Theorem 3.1.1.

3.4 Approximating Knapsack

3.4.1 Known lemmas

By known reductions (e.g., [11, 26]), we can focus on solving the following cleaner problem, which already captures the main difficulty of knapsack.

Problem 2. *Assume $\varepsilon \in (0, 1/2)$ and $1/\varepsilon \in \mathbb{N}^+$. Given a list I of items $(p_1, w_1), \ldots, (p_n, w_n)$ with weights $w_i \in \mathbb{N}$ and profits p_i being multiples of ε in the interval $[1, 2)$, compute a profit function that $(1-\varepsilon)$-approximates f_I up to $2/\varepsilon$.*

Lemma 3.4.1. *If for some $c \geq 2$, Problem 2 can be solved in $\widetilde{O}(n + 1/\varepsilon^c)$ time, then $(1-\varepsilon)$-approximating* Knapsack *can also be solved in $\widetilde{O}(n + 1/\varepsilon^c)$ time.*

Based on Chan's techniques [11], Jin [26] obtained the following lemmas for $(1-\varepsilon)$-approximating knapsack up to a small B or when there are few distinct values p_i.

Lemma 3.4.2 (Follows from Lemma 17 of [26])**.** *Given a list I of items $(p_1, w_1), \ldots, (p_n, w_n)$ with weights $w_i \in \mathbb{N}$ and profits p_i being multiples of ε in the interval $[1, 2)$, one can $(1-\varepsilon)$-approximate the profit function f_I up to B in $\widetilde{O}(n + \varepsilon^{-2}B^{1/3}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ time.*

Lemma 3.4.3 (Follows from Lemma 18 of [26])**.** *Given a list I of items $(p_1, w_1), \ldots, (p_n, w_n)$ with weights $w_i \in \mathbb{N}$ and profits p_i being multiples of ε in the interval $[1, 2)$, if there are only*

have m distinct profit values p_i, then one can $(1-\varepsilon)$-approximate the profit function f_I in $\widetilde{O}(n+\varepsilon^{-3/2}m^{3/4}/2^{\Omega(\sqrt{\log(1/\varepsilon)}})$ time.

The following useful lemma allows us to merge multiple profit functions, which was proved by Chan using divide-and-conquer and improved algorithms (min, +)-convolution [6, 45, 12].

Lemma 3.4.4 ([11, Lemma 2(i)])**.** *Let $f_1,\ldots,f_m$ be monotone step functions with total complexity $O(n)$ and ranges contained in $\{0\}\cup[A,B]$. Then we can compute a monotone step function that has complexity $\widetilde{O}(\frac{1}{\varepsilon}\log B/A)$ and $(1-O(\varepsilon))$-approximates $f_1\oplus\cdots\oplus f_m$, in $O(n)+\widetilde{O}((\frac{1}{\varepsilon})^2 m/2^{\Omega(\sqrt{\log(1/\varepsilon)})}\log B/A)$ time.*

3.4.2 Reduction Based on Greedy Exchange Argument

The goal of this section is to prove the following Lemma 3.4.5. Our algorithm is based on a greedy exchange argument similar to [26, Lemma 20], but we can obtain better bounds by combining with number theoretic results on dense subset sums.

Lemma 3.4.5. *Given a list I of n items with p_i being multiples of ε in interval $[1,2)$, and integer $1\le m\le n$ with $m=O(1/\varepsilon)$, one can compute in $\widetilde{O}(n+\varepsilon^{-11/5}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ time a profit function that $(m\varepsilon)$-additively approximates f_I up to $2m$.*

The proof of Lemma 3.4.5 assumes the following ingredient, which will be proved in later sections using random partitioning.

Lemma 3.4.6. *Given a list I of $n=O(1/\varepsilon)$ items with p_i being multiples of ε in interval $[1,2)$, one can compute in $O(n^{4/5}\varepsilon^{-7/5}/2^{\Omega(\sqrt{\log(1/\varepsilon)}})$ time a profit function that $(n\varepsilon)$-additively approximates f_I.*

Now we proceed to describe the algorithm for Lemma 3.4.5. Given items $(p_1,w_1),\ldots,(p_n,w_n)$, where $p_i\in[1,2)$ are multiples of ε, we sort them by non-increasing order of efficiency, $p_1/w_1\ge p_2/w_2\ge\cdots\ge p_n/w_n$. Then, we consider prefixes of this sequence of items, and define the following measure of diversity:

Definition 3.4.7 ($D(i)$)**.** For $1 \le i \le n$, let $D(i) = \min_J C([i] \setminus J)$, where the minimization is over all subsets $J \subseteq [i]$ with $|J| \le 2m$, and $C([i] \setminus J)$ denote the number of distinct values in $\{p_j : j \in [i] \setminus J\}$.

We have the following immediate observations about $D(i)$.

Observation 3.4.8. *1. For all $2 \le i \le n$, $0 \le D(i) - D(i-1) \le 1$.*

2. $D(i)$ (and the minimizer J) can be computed in $\widetilde{O}(i)$ time by the following greedy algorithm: Start with all values $p_1, p_2, \ldots, p_i$. Repeat the following up to $2m$ times: remove the value p_j with the minimum multiplicity, and add j into J.

Now, we set parameter $\Delta = \lfloor \varepsilon^{-10/13} \rfloor$. Define $i \in \{1, 2, \ldots, n\}$ be the maximum such that $D(i) \le \Delta$, which can be found using Observation 3.4.8 with a binary search in $\widetilde{O}(n)$ time.

The following lemma is the key component in our proof of Lemma 3.4.5.

Lemma 3.4.9 (Greedy Exchange Lemma)**.** *Let $S \subseteq [n]$ be any item set with total profit $\sum_{s \in S} p_s \le 2m$. Let $B := 9c\varepsilon^{-1}/\Delta$, where $c \ge 1$ is the universal constant in Lemma 3.2.6.*

Then, there exists an item set $\widetilde{S} \subseteq [n]$, such that the total profit $\widetilde{p}$ contributed by items $[n] \setminus [i]$ in $\widetilde{S}$ satisfies

$$\widetilde{p} := \sum_{s \in \widetilde{S} \cap ([n] \setminus [i])} p_s \le B, \tag{3.3}$$

and

$$\sum_{s \in \widetilde{S}} p_s \ge (1 - \varepsilon) \sum_{s \in S} p_s, \tag{3.4}$$

and

$$\sum_{s \in \widetilde{S}} w_s \le \sum_{s \in S} w_s. \tag{3.5}$$

Proof. If $D(i) < \Delta$, then by the definition of i we have $i = n$, and we can simply let $\widetilde{S} = S$, since $\widetilde{p} = 0$ always holds. So in the following we assume $D(i) = \Delta$.

We define $\widetilde{S} \subseteq [n]$ as the maximizer of

$$\sum_{s\in\widetilde{S}\cap[i]} p_s + \sum_{s\in\widetilde{S}\cap([n]\setminus[i])} (1-\varepsilon)p_s$$

among all $\widetilde{S}$ satisfying $\sum_{s\in\widetilde{S}} w_s \le \sum_{s\in S} w_s$. We claim that $\widetilde{S}$ satisfies the properties (3.3), (3.4), (3.5). Observe that (3.4), (3.5) immediately follow from the definition of $\widetilde{S}$. The main part is to verify (3.3).

Suppose for contradiction that (3.3) does not hold. Then, we can find a subset $K \subseteq \widetilde{S} \cap ([n] \setminus [i])$ with total profit $p^* = \sum_{k\in K} p_k \in (B, B+2]$, which can be obtained by removing items from $\widetilde{S} \cap ([n] \setminus [i])$ (recall that each item has profit in $[1, 2)$).

Define item set $I' := [i] \setminus \widetilde{S}$. Since $|\widetilde{S}| < \sum_{s\in\widetilde{S}} p_s / \min_{s\in\widetilde{S}} p_s \le \sum_{s\in\widetilde{S}} p_s \le 2m$, by the definition of $D(i)$, we know that $\{p_i : i \in I'\}$ contains at least $D(i) = \Delta$ distinct elements.

We apply Lemma 3.2.6 on the set of integers $X = \{p_i/\varepsilon : i \in I'\} \subseteq [1/\varepsilon, 2/\varepsilon)$ which contains at least Δ distinct integers, where the premise $1/\varepsilon = o(\Delta^2/\log\Delta)$ in Lemma 3.2.6 is satisified by our choice of $\Delta = \lfloor \varepsilon^{-10/13} \rfloor$. Lemma 3.2.6 states that for every $t \in [c\varepsilon^{-2}/\Delta, 0.5\Delta/\varepsilon]$, there exists $t' \in \mathcal{S}(X)$ such that $0 \le t' - t \le 8\varepsilon^{-1}/\Delta$. Here we set

$$t := \frac{(1-\varepsilon)p^*}{\varepsilon} + \frac{\varepsilon^{-1}}{\Delta},$$

which satisfies $t > p^*(1-\varepsilon)/\varepsilon > (1-\varepsilon)B/\varepsilon > c\varepsilon^{-2}/\Delta$, and $t < (B+2)/\varepsilon + \varepsilon^{-1}/\Delta = 9c\varepsilon^{-2}/\Delta + 2/\varepsilon + \varepsilon^{-1}/\Delta \le O(\varepsilon^{-16/13}) \le 0.5\Delta/\varepsilon$. Then the conclusion of Lemma 3.2.6 says that there is a subset $R \subseteq I'$ of items with total profit $\widetilde{p} := \varepsilon \cdot t'$, satisfying

$$1/\Delta \le \widetilde{p} - p^*(1-\varepsilon) \le 9/\Delta. \tag{3.6}$$

Note that (3.6) implies

$$
\begin{aligned}
p^* - \widetilde{p} &\geq \varepsilon \cdot p^* - 9/\Delta \\
&> \varepsilon \cdot B - 9/\Delta \\
&= \varepsilon \cdot 9c\varepsilon^{-1}/\Delta - 9/\Delta \\
&\geq 0.
\end{aligned}
$$

Recall that $R \subseteq I' = [i] \setminus \widetilde{S}$ and $K \subseteq \widetilde{S} \cap ([n] \setminus [i])$, which must both be non-empty. Since the efficiency of items are sorted in non-increasing order, we have $\min_{r \in R} p_r/w_r \geq \max_{k \in K} p_k/w_k$. Now we define the set of items

$$
\widetilde{S}' := (\widetilde{S} \setminus K) \cup R.
$$

Then, we have

$$
\begin{aligned}
\sum_{s \in \widetilde{S}} w_s - \sum_{s \in \widetilde{S}'} w_s &= \sum_{k \in K} w_k - \sum_{r \in R} w_r \\
&\geq \frac{\sum_{k \in K} p_k}{\max_{k \in K}(p_k/w_k)} - \frac{\sum_{r \in R} p_r}{\min_{r \in R}(p_r/w_r)} \\
&\geq \frac{1}{\min_{r \in R}(p_r/w_r)} \cdot \left(\sum_{k \in K} p_k - \sum_{r \in K} p_r \right) \\
&= \frac{1}{\min_{r \in R}(p_r/w_r)} \cdot (p^* - \widetilde{p}) \\
&\geq 0.
\end{aligned}
$$

Hence, $\sum_{s\in\widetilde{S}'} w_s \le \sum_{s\in\widetilde{S}} w_s$. On the other hand, by (3.6), we know that

$$\begin{aligned}
&\left(\sum_{s\in\widetilde{S}'\cap[i]} p_s + \sum_{s\in\widetilde{S}'\cap([n]\setminus[i])} (1-\varepsilon)p_s\right) - \left(\sum_{s\in\widetilde{S}\cap[i]} p_s + \sum_{s\in\widetilde{S}\cap([n]\setminus[i])} (1-\varepsilon)p_s\right)\\
&= \sum_{r\in R} p_r - \sum_{k\in K}(1-\varepsilon)p_k\\
&= \widetilde{p} - (1-\varepsilon)p^*\\
&> 0,
\end{aligned}$$

contradicting the definition of $\widetilde{S}$ being a maximizer.

Hence, we have established that $\widetilde{S}$ satisfies (3.3). □

Now we are ready to prove Lemma 3.4.5.

Proof of Lemma 3.4.5. Recall that $i \in \{1, 2, \ldots, n\}$ is the maximum such that $D(i) \le \Delta$, which can be found using Observation 3.4.8 with a binary search in $\widetilde{O}(n)$ time. Let $J \subset [i]$ with $|J| \le 2m$ be the minimizer for $D(i)$.

Now, we approximately compute the profit functions $f_J, f_{[i]\setminus J}, f_{[n]\setminus[i]}$ for three item sets $J, [i] \setminus J, [n] \setminus [i]$ using different algorithms, described as follows:

1. Use Lemma 3.4.6 to compute f_1 that $(2m\varepsilon)$-additively approximates f_J, in $O(m^{\frac{4}{5}}\varepsilon^{-\frac{7}{5}}/2^{\Omega(\sqrt{\log(1/\varepsilon)}}) \le O(\varepsilon^{-\frac{11}{5}}/2^{\Omega(\sqrt{\log(1/\varepsilon)}})$ time.

2. By definition of i, items in $[i] \setminus J$ have no more than Δ distinct profit values. Hence we can use Lemma 3.4.3 to compute f_2 that $(1-\varepsilon)$-approximates $f_{[i]\setminus J}$, in $\widetilde{O}(\Delta^{3/4}\varepsilon^{-3/2}) = \widetilde{O}(\varepsilon^{-27/13})$ time.

3. Use Lemma 3.4.2 to compute f_3 that $(1-\varepsilon)$-approximates the $f_{[n]\setminus[i]}$ up to $B = \Theta(\varepsilon^{-1}/\Delta)$ (defined in Lemma 3.4.9), in $\widetilde{O}(B^{1/3}\varepsilon^{-2}) \le \widetilde{O}(\varepsilon^{-27/13})$ time.

Finally, merge the three parts f_1, f_2, f_3 using Lemma 3.4.4 in $\widetilde{O}(\varepsilon^{-2})$ time, and return the result.

In the third part, the correctness of only computing up to B is justified by Lemma 3.4.9, which shows that if we only consider approximating sets with total profit up to $2m$, then we can assume the items in $[n] \setminus [i]$ only contributes profit at most B (3.3), at the cost of only incurring an $(1 - \varepsilon)$ approximation factor (3.4).

To analyze the error, notice that in the first part we incur an additive error of $(2m\varepsilon)$. In the second and third part and the final merging step we incur $(1 - O(\varepsilon))$ multiplicative error, which turns into $O(m\varepsilon)$ additive error since we only care about approximating up to $2m$. Hence the overall additive error is $O(m\varepsilon)$, which can be made $m\varepsilon$ by lowering the value of ε. $\square$

Now we show that Lemma 3.4.5 can be used to solve Problem 2, which is sufficient for proving Theorem 3.1.2.

Proof of Theorem 3.1.2. To solve Problem 2, we divide $[1, 2\varepsilon^{-1})$ into $O(\log(1/\varepsilon))$ many intervals $[m, 2m)$ where m are powers of 2, and use Lemma 3.4.5 to obtain profit functions achieving $m\varepsilon$-additive approximation up to $2m$. Then, taking their pointwise minimum yields an $(1 - O(\varepsilon))$ approximation. $\square$

In the following sections, we will prove Lemma 3.4.6.

3.4.3 Approximation using Δ-multiples of small set Δ

We first introduce several additional tools borrowed from previous works that will be used in our final proof of Lemma 3.4.6.

Following [11]'s terminology, we say a monotone step function is *p-uniform* if its function values are $0, p, 2p, \ldots, lp$ for some l. A p-uniform function is said to be *pseudo-concave*, if the differences of consecutive x-breakpoints are nondecreasing from left to right. An example of a p-uniform and pseudo-concave function is the profit function f_I of a set I of items with the same profit $p_i = p$, which can be exactly computed by simple greedy: the function f_I takes values $0, p, 2p, \ldots, np$, with x-breakpoints $w_1, w_1 + w_2, \ldots, w_1 + \cdots + w_n$, where w_i's are sorted in nondecreasing order.

As in [11] and [26], we will use the method of approximation via Δ-multiples. For a set Δ of numbers, we say that p is a Δ-multiple if it is a multiple of δ for some $\delta \in \Delta$. Chan [11] used the SMAWK algorithm [2] and suitable rounding to prove the following lemma:

Lemma 3.4.10 ([11, Lemma 5])**.** *Let $f_1, \ldots, f_m$ be monotone step functions with ranges contained in $[0, B]$. Let $\Delta \subset [\delta, 8\delta]$. If every f_i is p_i-uniform and pseudo-concave for some $p_i \in [1, 2]$ which is a Δ-multiple, then we can compute a monotone step function that $O(|\Delta|\delta)$-additively approximates $\min\{f_1 \oplus \cdots \oplus f_m, B\}$ in $\widetilde{O}(Bm/\delta)$ time.*

Chan [11] gave a construction of a small set Δ such that every real number in $[1, 2]$ can be approximated by a Δ-multiples. Here, we present a more simplified construction.

Lemma 3.4.11. *For parameters $0 < \varepsilon < \delta < 1/2$, let $r = \lceil \log_{1+\varepsilon}(1 + 2\delta) \rceil = O(\delta/\varepsilon)$, and define $a_i = \delta(1+\varepsilon)^i$ for $0 \le i \le r+1$. Let $\Delta = \{a_i\}$ be the set of a_i. Then for any $t \in [1, 2]$, there is a multiple of some a_i in the range $[t, t + 2\varepsilon]$. Thus, every real number in $[1, 2]$ could be approximated by a Δ multiple with $O(\varepsilon)$ additive error, where $|\Delta| = r + 2 = O(\delta/\varepsilon)$ and all elements in Δ are within $[\delta, 8\delta]$.*

Proof. Let c be the largest integer such that $(t + 2\varepsilon)/c \ge a_0$. Since c is largest, $c + 1 \ge (t + 2\varepsilon)/a_0 \ge 1/\delta$, so $(c+1)/c \le (1/\delta)/(1/\delta - 1) = 1 + \delta/(1 - \delta) \le 1 + 2\delta \le a_{r+1}/a_0$. Since $(t + 2\varepsilon)/(c + 1) < a_0$, we know $(t + 2\varepsilon)/c < a_{r+1}$. Let k be the largest integer in $[0, r]$ such that $a_k \le (t + 2\varepsilon)/c$. Note $a_{k+1} > (t + 2\varepsilon)/c$, so $a_k = a_{k+1}/(1 + \varepsilon) \ge t/c$ using the fact that $t \le 2$. As a result, $a_k \in [t/c, (t + 2\varepsilon)/c]$, thus $ca_k \in [t, t + 2\varepsilon]$. $\square$

3.4.4 Random Partitioning

Assume that $n < 1/\varepsilon$. In the section, we will use random partitioning to prove Lemma 3.4.6, restated below.

Lemma 3.4.6. *Given a list I of $n = O(1/\varepsilon)$ items with p_i being multiples of ε in interval $[1, 2)$, one can compute in $O(n^{4/5}\varepsilon^{-7/5}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ time a profit function that $(n\varepsilon)$-additively approximates f_I.*

Proof. Set $\Delta_1 = \Theta(\sqrt{n})$ and $\Delta_0 = \Theta(n^{\frac{7}{10}}\varepsilon^{\frac{2}{5}}2^{c\sqrt{\log(1/\varepsilon)}})$ for some small constant $c > 0$. Assume that Δ_0 is a power of 2 without loss of generality. Note that $\Delta_0 = O(\Delta_1)$, which follows from $n = O(1/\varepsilon)$.

Claim 3.4.12. *We can partition elements of I into $\Theta(\Delta_1)$ groups $G_1, G_2, \ldots, G_k$, each of size $O(n/\Delta_1)$, while all elements within group G_i are multiples of p_i for some $p_i = \Theta(\Delta_1\varepsilon)$.*

Proof. In Lemma 3.4.11, plugging in $\delta = \varepsilon\Delta_1$, we obtain a set A of size $O(\Delta_1)$ whose elements are of order $\Theta(\Delta_1\varepsilon)$, and each item in I can be $(1+\varepsilon)$-approximated by A-multiples.

We group the elements in I by their divisor in A. We then evenly split groups with size more than n/Δ_1 into two until all groups have sizes of at most n/Δ_1. □

From now on, assume that $G_1, G_2, \ldots, G_k$ are groups satisfying conditions in Claim 3.4.12.

We now randomly partition $\{1, 2, \ldots, k\}$ into Δ_0 parts, $I_1, \ldots, I_{\Delta_0}$, by assigning each $1 \le i \le k$ into some $I_j (1 \le j \le \Delta_0)$ independently and uniformly. Then, set $X_j = \bigcup_{i \in I_j} G_i$ for every $1 \le j \le \Delta_0$. It is easy to see $\{X_j\}$ is a partition of I.

Claim 3.4.13. *With probability $\ge 3/4$, $|I_j| = O(\Delta_1/\Delta_0)$, and hence $|X_j| \le O(n/\Delta_0)$.*

Proof. By Chernoff bound[1], for some large constant $c > 0$, $|I_j| \ge ck/\Delta_0$ happens with probability at most $1/(4\Delta_0)$. Thus $|I_j| = O(\Delta_1/\Delta_0)$ holds for all j with probability $\ge 3/4$ by union bound. By Claim 3.4.12, $|X_j| \le |I_j| O(n/\Delta_1) = O(m/\Delta_0)$. □

Now assume the event in Claim 3.4.13 happens.

Claim 3.4.14. *We can approximate $\bigoplus_{x \in X_j} f_x$ with additive error $O(n\varepsilon/\Delta_0)$ for all $1 \le j \le \Delta_0$ in*
$\widetilde{O}(n^2\varepsilon^{-1}/(\Delta_0\Delta_1)) = \widetilde{O}(n^{4/5}\varepsilon^{-7/5}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$ *time.*

Proof. Fix a single j. By Claim 3.4.13, $\bigoplus_{x \in X_j} f_x$ is the convolution of $O(n/\Delta_0)$ elements, each being a multiple of order $\Theta(\Delta_1\varepsilon)$. By applying Lemma 3.4.10 with parameters $B =$

[1]For independent random variables $x_1, \ldots, x_n \in \{0, 1\}$ and $\delta > 0, 0 \le w_1, \ldots, w_n \le 1$, let $X = \sum_{i=1}^n w_i x_i$ and $\mu = \mathbb{E}[x]$, then $\mathbf{Pr}[|x - \mu| \ge \delta\mu] \le 2e^{-\delta^2\mu/3}$

$O(n/\Delta_0), \delta = \Theta(\Delta_1\varepsilon), |\Delta| = |I_j| = O(\Delta_1/\Delta_0)$, we can approximate $\bigoplus_{x\in X_j} f_x$ with additive error $O(\Delta_1^2\varepsilon/\Delta_0) = O(n\varepsilon/\Delta_0)$ within time $\widetilde{O}((n/\Delta_0)^2/(\Delta_1\varepsilon))$.

We can do so for all $1 \le j \le \Delta_0$, with running time $\widetilde{O}(n^2/(\Delta_0\Delta_1\varepsilon)) = \widetilde{O}(n^{4/5}\varepsilon^{-7/5}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$.

□

Now we can replace $\bigoplus_{x\in X_j} f_x$ by the approximation obtained in Claim 3.4.14, since the total additive error inflicted will be $O(n\varepsilon/\Delta_0)\Delta_0 = O(n\varepsilon)$.

We use divide and conquer to combine the answer of $\bigoplus_{x\in X_j} f_x$. The merge process can be viewed as a complete binary tree with Δ_0 leaves. For $S \subseteq \{1, 2, \dots, \Delta_0\}$, define $F(S) = \bigoplus_{x\in\cup_{(s\in S)}X_s} f_x$. Claim 3.4.14 allows us to approximate $F(S)$ for all $|S| = 1$. Now we have the following claim regarding combining two subtrees S_1 and S_2.

Claim 3.4.15. *Assume $i \le \log_2 \Delta_0$ and $|S_1| = |S_2| = 2^i$, where $S_1, S_2 \subseteq \{1, 2, \dots, \Delta_0\}$ and $S_1 \cap S_2 = \emptyset$. Assume that A_1 is an approximation of $F(S_1)$ with additive error err_1, A_2 is an approximation of $F(S_2)$ with additive error err_2. Then with probability $\ge 1 - 1/(5\Delta_0)$, we can compute an approximation of $F(S_1 \cup S_2)$ with additive error $err_1 + err_2 + O(2^{0.9i}n\varepsilon/\Delta_0)$ in time $O(\varepsilon^{-2}\Delta_0^{0.5}/(\Delta_1^{0.5}2^{\Theta(\sqrt{\log(1/\varepsilon)})}))$.*

Proof. Define $\delta_i = 2^{0.9i}n\varepsilon/\Delta_0$. A naive way to approximate $F(S_1 \cup S_2)$ is to round each value in A_1 and A_2 to a multiple of δ_i, and then invoke the $(\min, +)$ convolution as in Lemma 3.4.4. In the following we will show a better method exploiting the fact that $\{X_i\}$ is a random partition.

Let the global optimal solution be to choose the subset T of items. Define $H_1 = \bigcup_{i\in S_1} X_i, H_2 = \bigcup_{i\in S_2} X_i$. Note that the groups $G_1, \dots, G_k$ are assigned into $X_1, \dots, X_{\Delta_0}$ uniformly randomly. Pick $u = Cn\sqrt{2^i/(\Delta_1\Delta_0)}\log n$ for a large constant $C > 0$. By Chernoff bound, the probability that $\mathbf{Pr}(|\sum_{x\in T\cap H_1} x - \sum_{x\in T\cap H_2} x| \ge u) \le 1/(5n)$[2].

[2]We apply Chernoff bound with $w_j = R_j/(2n/\Delta_1)$ where $R_j = \sum_{x\in T\cap G_j} x$. Now consider S_1. We set $x_j = 1$ if $j \in \cup_{t\in S_1} I_t$ and $x_j = 0$ otherwise. Since the partition $\{I_t\}_{1\le t\le\Delta_0}$ is random, the expected value of $\sum_{j=1}^k w_j x_j$ will be $\Theta(\Delta_1 2^i/\Delta_0)$. From Chernoff bound, this value will be $u/(4n/\Delta_1)$ away from expected value with probability $\le 2e^{\Theta(-(u/(4n/\Delta_1))^2/(\Delta_1 2^i/\Delta_0)))} \le 1/(10n)$. By union bound, both $\sum_{x\in T\cap H_2} x$ and $\sum_{x\in T\cap H_1} x$ will be within difference $u/2$ from the expected value with probability $\ge 1 - 1/(5n)$, in which case their difference will be bounded by u.

Now assume that $|\sum_{x\in T\cap H_1} x - \sum_{x\in T\cap H_2} x| \le u$, and we show how to approximate $F(S_1 \cup S_2)$ under the assumption. During the $(\min,+)$ convolution, we first round the values of $F(S_1), F(S_2)$ to multiples of δ_i. Then we only need to consider the pairs that differ in value by at most u. We then divide the arrays into blocks with values within a difference of u from each other, and do $(\min,+)$-convolution between the pairs of blocks with indices differing by at most 1. The block sizes are at most $u/\delta_i = \widetilde{O}(\varepsilon^{-1}\Delta_0^{0.5}\Delta_1^{-0.5}/2^{0.4i})$, so the running time for each $(\min,+)$-convolution is $O(\varepsilon^{-2}\Delta_0/(\Delta_1 2^{0.8i}2^{\Omega(\sqrt{\log(1/\varepsilon)})}))$ using Williams's $O(n^2/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$-time algorithm for length-n $(\min,+)$-convolution [45]. Since the value in the merged answer is bounded by $\widetilde{O}(2^i n/\Delta_0)$ by Claim 3.4.13, there are $\widetilde{O}(2^i n/(\Delta_0 u))$ min-plus convolutions in total, with total complexity $\widetilde{O}(\varepsilon^{-2}2^i n/(\Delta_1 2^{0.8i}2^{\Omega(\sqrt{\log(1/\varepsilon)})}u)) = O(\varepsilon^{-2}\Delta_0^{0.5}/(\Delta_1^{0.5}2^{\Theta(\sqrt{\log(1/\varepsilon)})}))$. □

Now we conclude the proof by applying Claim 3.4.15 to the divide and conquer process. Assume all the $\le \Delta_0$ many calls to Claim 3.4.15 yield correct approximations, which happens with success probability $\ge 3/4$ by union bound.

To analyze the error term, note that there are $O(\Delta_0/2^i)$ merges of two subtrees with 2^i parts each, where Claim 3.4.15 inflicts additive error $O(2^{0.9i}n\varepsilon/\Delta_0)$ for each such a merge. Thus the total additive error is bounded by $\sum_{2^i \le \Delta_0}(2^{0.9i}n\varepsilon/\Delta_0)(\Delta_0/2^i) = O(n\varepsilon)$.

Now we analyze the time complexity. Note that the total complexity for the i-th layer is

$$\begin{aligned}
&O((\Delta_0/2^i)\cdot \varepsilon^{-2}\Delta_0^{0.5}\Delta_1^{-0.5}/2^{\Omega(\sqrt{\log(1/\varepsilon)}})\\
=\,&O(\varepsilon^{-2}\Delta_0^{1.5}\Delta_1^{-0.5}/2^{\Omega(\sqrt{\log(1/\varepsilon)}})\\
=\,&O(n^{\frac{4}{5}}\varepsilon^{-\frac{7}{5}}/2^{\Omega(\sqrt{\log(1/\varepsilon)}}).
\end{aligned}$$

As there are logarithmically many layers, the total complexity for the divide and conquer part is $O(n^{\frac{4}{5}}\varepsilon^{-\frac{7}{5}}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$.

Thus our total complexity is $O(n^{\frac{4}{5}}\varepsilon^{-\frac{7}{5}}/2^{\Omega(\sqrt{\log(1/\varepsilon)})})$, and with a success probability of $\ge 1/2$ (which can be amplified by repetition) by union bound.

A small detail is that when n is so small that $n^{7/10} < \varepsilon^{-2/5}$, $\Delta_0 < 1$ and our reasoning

falls apart. In such cases, one can simply set $\Delta_0 = 1$ and the running time still holds. $\square$

Chapter 4

Concluding Remarks

In this thesis, I have shown three fine-grained upper bounds based on the dynamic programming technique. All three final algorithms have several things in common:

- One or several dynamic programming schemes lie at the core of the algorithm.
- Some advanced approach (Matrix Multiplication or FFT) is used to speed up some computation in the dynamic programming schemes.
- Some approaches (combinatorial or number theoretical), often complicated, are used to prune useless computation or to re-order different stages of computation for efficiency.
- There (likely) **do not** to achieve the theoretical lower bound for the corresponding problem.

At the end of this thesis, I want to put emphasis on what DP **did not** do, namely that it (is likely to have) failed to achieve the theoretical lower bound for the corresponding problems: it is highly unlikely that the best exact algorithm for unweighted TED runs in $O(n^{2.9\cdots})$ time, the best FPTAS for Partition runs in $\widetilde{O}(n + \varepsilon^{-1.25})$ time (it is in fact believed to be $\widetilde{O}(n+\varepsilon^{-1})$), or the best FPTAS for Knapsack runs in $\widetilde{O}(n+\varepsilon^{-2.2})$ time (it is in fact believed to be $\widetilde{O}(n + \varepsilon^{-2})$). Moreover, even to achieve the running time we currently have, we have to combine DP with many other advanced and complicated techniques. These suggest that

perhaps DP is not the final answer — perhaps an alternative to DP still remains elusive. Such technique, if it does exist, could yield simple and elegant solutions to many fundamental problems and revolutionize the field. But does this alternative approach really exist, or do we simply need to try harder on improving existing approaches? Only time will tell.

Bibliography

[1] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 41–57, 2019.

[2] Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter Shor, and Robert Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1):195–208, November 1987.

[3] John Bellando and Ravi Kothari. Region-based modeling and tree edit distance as a basis for gesture recognition. In *Proceedings of the 10th International Conference on Image Analysis and Processing (ICIAP)*, pages 698–703. IEEE Computer Society, 1999.

[4] Richard E Blahut. *Fast algorithms for signal processing*. Cambridge University Press, 2010.

[5] Mahdi Boroujeni, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, and Saeed Seddighin. $1+\varepsilon$ approximation of tree edit distance in quadratic time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 709–720, 2019.

[6] David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Pătraşcu, and Perouz Taslakian. Necklaces, convolutions, and x+y. *Algorithmica*, 69(2):294–314, June 2014.

[7] 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 *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 375–384, 2016.

[8] Karl Bringmann and Vasileios Nakos. A fine-grained perspective on approximating subset sum and partition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1797–1815. SIAM, 2021.

[9] Karl Bringmann and Philip Wellnitz. On near-linear-time algorithms for dense subset sum. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on*

Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021, pages 1777–1796. SIAM, 2021.

[10] Peter Buneman, Martin Grohe, and Christoph Koch. Path queries on compressed XML. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pages 141–152, 2003.

[11] Timothy M. Chan. Approximation Schemes for 0-1 Knapsack. In *Proceedings of the 1st Symposium on Simplicity in Algorithms (SOSA)*, pages 5:1–5:12, 2018.

[12] Timothy M. Chan and Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1246–1255, 2016.

[13] Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, pages 90–101, 1999.

[14] Weimin Chen. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40(2):135–158, 2001.

[15] Shucheng Chi, Ran Duan, and Tianle Xie. *Faster Algorithms for Bounded-Difference Min-Plus Product*, pages 1435–1447.

[16] Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. On problems equivalent to (min,+)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, January 2019.

[17] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. In Lars Arge, Christian Cachin, Tomasz Jurdziński, and Andrzej Tarlecki, editors, *Automata, Languages and Programming*, pages 146–157, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[18] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1):4:1–4:33, 2009.

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

[20] George Gens and Eugene Levner. Computational complexity of approximation algorithms for combinatorial problems. In Jirí Becvár, editor, *Mathematical Foundations of Computer Science 1979, Proceedings, 8th Symposium, Olomouc, Czechoslovakia, September 3-7, 1979*, volume 74 of *Lecture Notes in Computer Science*, pages 292–300. Springer, 1979.

[21] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[22] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[23] Matthias Höchsmann, Thomas Töller, Robert Giegerich, and Stefan Kurtz. Local similarity in RNA secondary structures. In *Proceedings of 2nd IEEE Computer Society Bioinformatics Conference, CSB*, pages 159–168. IEEE Computer Society, 2003.

[24] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM (JACM)*, 22(4):463–468, October 1975.

[25] Klaus Jansen and Stefan E.J. Kraft. A faster fptas for the unbounded knapsack problem. *European Journal of Combinatorics*, 68:148 – 174, 2018.

[26] Ce Jin. An improved FPTAS for 0-1 knapsack. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 76:1–76:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[27] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[28] Hans Kellerer, Renata Mansini, Ulrich Pferschy, and Maria Grazia Speranza. An efficient fully polynomial approximation scheme for the subset-sum problem. *J. Comput. Syst. Sci.*, 66(2):349–370, 2003.

[29] Hans Kellerer and Ulrich Pferschy. A new fully polynomial time approximation scheme for the knapsack problem. *Journal of Combinatorial Optimization*, 3(1):59–71, July 1999.

[30] Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an fptas for the knapsack problem. *Journal of Combinatorial Optimization*, 8(1):5–11, March 2004.

[31] Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium on Algorithms (ESA)*, volume 1461 of *Lecture Notes in Computer Science*, pages 91–102. Springer, 1998.

[32] Philip N. Klein, Thomas B. Sebastian, and Benjamin B. Kimia. Shape matching using edit-distance: an implementation. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms (SODA)*, pages 781–790, 2001.

[33] Philip N. Klein, Srikanta Tirthapura, Daniel Sharvit, and Benjamin B. Kimia. A tree-edit-distance algorithm for comparing simple, closed shapes. In David B. Shmoys, editor, *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 696–704, 2000.

[34] Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 21:1–21:15, 2017.

[35] Eugene L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.

[36] Vsevolod F Lev. Blocks and progressions in subset sum sets. *ACTA ARITHMETICA-WARSZAWA-*, 106(2):123–142, 2003.

[37] Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. A subquadratic approximation scheme for partition. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 70–88, 2019. Full version at `https://arxiv.org/abs/1804.02269v2`.

[38] Donguk Rhee. Faster fully polynomial approximation schemes for knapsack problems. Master's thesis, Massachusetts Institute of Technology, 2015.

[39] A Sárkozy. Fine addition theorems, ii. *Journal of Number Theory*, 48(2):197–218, 1994.

[40] Thomas B. Sebastian, Philip N. Klein, and Benjamin B. Kimia. Recognition of shapes by editing their shock graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(5):550–571, 2004.

[41] Stanley M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977.

[42] Bruce A. Shapiro and Kaizhong Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Bioinformatics*, 6(4):309–318, 10 1990.

[43] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[44] Michael S. Waterman. *Introduction to computational biology: maps, sequences and genomes*. CRC Press, 1995.

[45] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 664–673, 2014.

[46] Kaizhong Zhang and Dennis E. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.