More on Change-Making and Related Problems

Timothy M. Chan
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
tmc@illinois.edu

Qizheng He
Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
qizheng6@illinois.edu

Abstract

Given a set of n integer-valued coin types and a target value t, the well-known *change-making* problem asks for the minimum number of coins that sum to t, assuming an unlimited number of coins in each type. In the more general *all-targets* version of the problem, we want the minimum number of coins summing to j, for every $j = 0, \ldots, t$. For example, the textbook dynamic programming algorithms can solve the all-targets problem in $O(nt)$ time. Recently, Chan and He (SOSA'20) described a number of $O(t \operatorname{polylog} t)$-time algorithms for the original (single-target) version of the change-making problem, but not the all-targets version.

In this paper, we obtain a number of new results on change-making and related problems:

- We present a new algorithm for the all-targets change-making problem with running time $\tilde{O}(t^{4/3})$, improving a previous $\tilde{O}(t^{3/2})$-time algorithm.
- We present a very simple $\tilde{O}(u^2 + t)$-time algorithm for the all-targets change-making problem, where u denotes the maximum coin value. The analysis of the algorithm uses a theorem of Erdős and Graham (1972) on the Frobenius problem. This algorithm can be extended to solve the all-capacities version of the *unbounded knapsack* problem (for integer item weights bounded by u).
- For the original (single-target) coin changing problem, we describe a simple modification of one of Chan and He's algorithms that runs in $\tilde{O}(u)$ time (instead of $\tilde{O}(t)$).
- For the original (single-capacity) unbounded knapsack problem, we describe a simple algorithm that runs in $\tilde{O}(nu)$ time, improving previous near-u^2-time algorithms.
- We also observe how one of our ideas implies a new result on the *minimum word break* problem, an optimization version of a string problem studied by Bringmann et al. (FOCS'17), generalizing change-making (which corresponds to the unary special case).

1 Introduction

In the *change-making* problem (also known as *coin changing*), a set of n positive-integer-valued coin types is given, and the cashier wants to use the minimum number of coins to sum to a target value t exactly, where the number of coins in each type can be used an unlimited number of times. This is a well-known textbook problem, which is weakly NP-hard [20], and standard solutions using dynamic programming [27] have $O(nt)$ running time.

Change-making is closely related to another textbook problem, *subset sum* (the differences are that in subset sum, each item may be used at most once and there is no objective function to minimize). A series of work in the last few years [5, 16, 15, 17] have given improved algorithms for subset sum, using convolution (FFT). Very recently, at SOSA'20, Chan and He [8] revisited the change-making problem and described a number of $O(t \operatorname{polylog} t)$-time algorithms, using FFT; their fastest deterministic and randomized algorithms have $O(t \log t \log \log t)$ and $O(t \log t)$ running time respectively.

All-targets change-making.   In this paper, we consider a more general, *all-targets* version of the change-making problem: the aim is to compute, for each target value $j = 0, \ldots, t$, the minimum number of coins that can be used to sum to j exactly. This version of the problem is equally natural. For instance, the standard $O(nt)$-time dynamic programming algorithms are actually designed to solve this more general version. Some of the newer subset-sum algorithms [5, 16, 17, 15] also solved the analogous all-targets version of subset sum, but in contrast, Chan and He's algorithms for change-making do *not* work for the all-targets version.

The best previous result for the all-targets change-making problem that we are aware of was an $\tilde{O}(t^{3/2})$-time[1] algorithm by Karl Bringmann and Tomasz Kociumaka (2019), cited as a personal communication (and briefly sketched) in a very recent paper by Lincoln, Polak, and Vassilevska Williams (ITCS'20) [19]. Lincoln et al.'s paper gave a web of fine-grained reductions connecting a variety of problems, including a reduction from all-targets change-making to the "monochromatic convolution" problem, the latter of which is shown to have near $n^{3/2}$ time complexity iff 3SUM has near quadratic time complexity. Their work implicitly hints at the possibility that the all-targets change-making problem might have near $t^{3/2}$ complexity as well, but the reduction is in the opposite direction.

Our first result is an $\tilde{O}(t^{4/3})$-time algorithm for the all-targets change-making problem, interestingly beating $t^{3/2}$ and placing the problem in a different category than monochromatic convolution and all its surrounding problems. Our algorithm is conceptually simple, exploiting an easy lemma on a binary special case of $(\min, +)$-convolution (using FFTs).

All-targets change-making in terms of u.   Next, we consider the complexity of the all-targets change-making problem in terms of some other natural parameters besides n and t: specifically,

- the largest coin value, denoted by u;
- the sum of the n given coin values, denoted by σ.

Some prior works have analyzed algorithms in terms of u and σ for the subset sum problem [22, 16]. A few recent papers have also analyzed algorithms in terms of u for the *0-1 knapsack* and the *unbounded knapsack* problem [2, 4, 12, 14, 24]. The unbounded knapsack problem is particularly relevant: given integer weights $w_1, \ldots, w_n$ and profits $p_1, \ldots, p_n$ and capacity value t, find nonnegative integers $m_1, \ldots, m_n$ to maximize $\sum_i m_i p_i$ such that $\sum_i m_i w_i \leq t$. Change-making is a special case, for example, by setting $w_i = v_i$ and $p_i = M v_i - 1$ for a sufficiently large M. Improving some previous algorithms [4, 24], Axiotis and Tzamos (ICALP'19) [2] and Jansen and Rohwedder (ITCS'19) [14] independently described algorithms[2] for unbounded knapsack running in $\tilde{O}(u^2)$ time with $u := \max_i w_i$ (the

---

[1] The $\tilde{O}$ notation hides polylogarithmic factors.

[2] We found that an $\tilde{O}(u^2)$ algorithm (basically the same as Axiotis and Tzamos') appeared earlier in a commentary on a 2016 programming contest problem by Arthur Nascimento, solved by Yan Soares Couto; see Problem L of `https://www.ime.usp.br/~maratona/assets/seletivas/2016/comentarios.pdf`.

time bound can be reduced slightly to $O(u^2/2^{\Theta(\sqrt{\log u})})$ by using known slightly subquadratic algorithms for $(\min, +)$-convolution [26]). However, these algorithms do not solve the all-targets or all-capacities version (computing the optimal profit for every capacity $j = 0, \ldots, t$).[3]

For the all-targets version of change-making, it is not difficult to obtain an $O(u^3 + t)$-time algorithm, based on a known observation that when the target is sufficiently large, it is always advantageous to use the largest coin. We describe a new algorithm that improves the running time to $O(u^2 \log u + t)$. Note that the algorithm is optimal for large $t \gg u^2 \log u$, since the output size for the all-targets problem is $\Omega(t)$.

The new algorithm is remarkably simple – just a slight variation of one of the standard dynamic programming solutions, with a 3-line pseudocode! (See page 8.) It is easily implementable and does not require FFT. However, the correctness argument is far from obvious, and requires a nice application of a number-theoretic theorem by Erdős and Graham [13] on the *Frobenius problem* (about the smallest target value that cannot be represented by a coin system). Arguably, algorithms that are simple but nontrivial to analyze are the most interesting kinds of algorithms.

All-capacities unbounded knapsack in terms of u. Our algorithm can be easily modified to solve the unbounded knapsack problem in the all-capacities version, with the same $O(u^2 \log u + t)$ time bound. This also implies an $O(u^2 \log u)$-time algorithm for the single-capacity version, which is a bit simpler than the previous $\tilde{O}(u^2)$ algorithms [2, 14] (in addition to extending it to all-capacities). For unbounded knapsack, a nearly matching conditional lower bound is known [9, 18]: more precisely, if single-capacity unbounded knapsack could be solved in truly subquadratic time for instances with $t, u = \Theta(n)$, then so could $(\min, +)$-convolution.

In terms of σ. We describe a variant of our algorithm with time bound $\tilde{O}((t\sigma)^{2/3} + t)$ for the all-targets change-making or all-capacities unbounded knapsack problem. Note that if $\sigma \ll t$, this is better than our earlier $\tilde{O}(t^{4/3})$ bound for the all-targets change-making.

Single-target change-making. For the single-target (original) change-making problem, we also describe how to improve the running time of one of Chan and He's FFT-based algorithms [8] from $\tilde{O}(t)$ to $\tilde{O}(u)$, which is faster than applying the previous $\tilde{O}(u^2)$-time algorithms [2, 14] for single-capacity unbounded knapsack.

Single-capacity unbounded knapsack. For the single-capacity (original) unbounded knapsack problem, we also describe a simple algorithm with running time $\tilde{O}(nu)$, which (ignoring $n^{o(1)}$ factors) simultaneously improves the standard $O(nt)$-time dynamic programming algorithm and the previous $\tilde{O}(u^2)$-time algorithms [2, 14] (since $u \leq t$ without loss of generality, and $n \leq u$ after pruning unnecessary items). There was a previous $O(nu)$-time algorithm by Pisinger [22] for subset sum, but not for unbounded knapsack.

Minimum word break. Finally, we consider a generalization of the problem for strings, known as the *minimum word break* problem: Given a string s with length n and a set D of strings (a "dictionary" of "words") with total length m, express s as a concatenation of words

---

[3] Cygan et al. [9] refered to the all-capacities version as UNBOUNDED-KNAPSACK$^+$; Kunnemann et al. [18] called it the *output-intensive* version.

from D, using the smallest number of words, where a word may be used multiple times. It is easy to see that if the alphabet is unary, then the problem is the same as change-making (the single-target version, with n and m corresponding to t and σ). A straightforward dynamic programming algorithm runs in $\tilde{O}(nd + m)$ time, where d denotes the number of distinct lengths among the words in D, by using randomized fingerprints [3] (which can be made deterministic [28]). Because $m \geq \frac{d(d+1)}{2}$, the bound is $\tilde{O}(n\sqrt{m} + m)$.

The decision version of the problem – deciding whether a solution exists, without minimizing the number of words – was considered by Bringmann, Grønlund, and Larsen (FOCS'17) [6], who gave an $\tilde{O}(nm^{1/3} + m)$-time algorithm, using FFT (improving a previous algorithm by Backurs and Indyk [3] with running time $\tilde{O}(nm^{1/2-1/18} + m)$). Bringmann et al. also proved a nearly matching conditional lower bound for combinatorial algorithms, assuming the conjecture that k-clique requires near n^k time for combinatorial algorithms. However, they did not obtain results on the *minimum* word break problem: part of the difficulty is that for the optimization problem, the various convolution operations needed change to $(\min, +)$-convolutions, which appear to be more expensive.

Nevertheless, we note that Bringmann et al.'s algorithm can still be adapted to solve the minimum word break problem. In fact, the time bound $\tilde{O}(nm^{1/3} + m)$ remains the same. This shows that surprisingly the optimization problem is not harder but has the same fine-grained complexity as the decision problem (at least for combinatorial algorithms, assuming the k-clique conjecture). The only new ingredient in our adaptation of Bringmann et al.'s algorithm is the same lemma on $(\min, +)$-convolutions that we have used in our $\tilde{O}(t^{4/3})$ algorithm for change-making.

2   Preliminaries

The all-targets version of the change-making problem can be formally defined as follows:

▶ **Problem 1** (ALL-TARGETS CHANGE-MAKING). *Given a set $V = \{v_1, \ldots, v_n\}$ of n positive integers (coin values) and an integer t, for each $j = 0, \ldots, t$, find the size of the smallest multiset S (duplicates allowed) of coin values from V such that S sums to exactly j, i.e., find the minimum m_j^* of $\sum_{i=1}^n m_i$ subject to the constraint that $\sum_{i=1}^n m_i v_i = j$, where $m_i \in \mathbb{N}$.*

Besides n (the number of coin values) and t (the maximum target value), we introduce two more parameters: let $u = \max_{i=1}^n v_i$ denote the maximum coin value, and $\sigma = \sum_{i=1}^n v_i$ denote the sum of input coin values. Simple observation reveals some inequalities relating the parameters: we have $n = O(\sqrt{\sigma})$ (because the distinctness of the v_i's implies $\sigma \geq \frac{n(n+1)}{2}$), $n \leq u$, $u \leq t$ (without loss of generality), and $\sigma \leq nu$. Note that unlike in the subset sum problem, t may be smaller or larger than σ.

Boolean convolution.   The Boolean convolution $A \circ B$ of two Boolean arrays $A[0, \ldots, t_1]$ and $B[0, \ldots, t_2]$ is a Boolean array with $t_1 + t_2 + 1$ elements, where $(A \circ B)[j] = \bigvee_{j'=0}^{t_1}(A[j'] \wedge B[j - j'])$ (we assume out-of-range values are 0).

Change-making is closely related with Boolean convolution. For any integer k, let $C_V^{(k)}[0, \ldots, t]$ denote the Boolean array where

$C_V^{(k)}[j] = 1$ iff there exist k coins from V with their sum being j.

Then $C_V^{(k)}$ can be obtained from the first $t + 1$ elements of $C_V^{(k_1)} \circ C_V^{(k_2)}$, for any $k_1, k_2 > 0$ where $k = k_1 + k_2$.

The Boolean convolution of two arrays of size $O(t)$ can be computed in $O(t \log t)$ time by FFT.

(min, +)-convolution.   The (min, +)-convolution $A \star B$ of two arrays $A[0, \ldots, t_1]$ and $B[0, \ldots, t_2]$ is an array with $t_1 + t_2 + 1$ elements, where $(A \star B)[j] = \min_{j'=0}^{t_1}(A[j'] + B[j - j'])$ (we assume out-of-range values are ∞).

Change-making is also related to (min, +)-convolution. For a set V of coin values, let $D_V[0, \ldots, t]$ denote the array where

$D_V[j]$ = the minimum number of coins from V needed to sum to j

(if no solution exists, $D_V[j] = \infty$). Then $D_{V_1 \cup V_2}$ can be obtained from the first $t + 1$ elements of $D_{V_1} \star D_{V_2}$.

It has been conjectured by some researchers that (min, +)-convolution cannot be solved in truly subquadratic time (e.g., see [9, 18]). However, the following lemma shows that a subquadratic algorithm is possible for the special case of (min, +)-convolution where the second array is "binary", i.e., all entries of B are in $\{1, \infty\}$. This "trick" is not new and is known before, for example, in the context of matrix multiplication (for computing the (min, +)-product when one of the matrices is binary [25, 11, 7]), with the basic idea tracing back to Matoušek's dominance algorithm [21].

▶ **Lemma 1.** *Given two arrays $A[0, \ldots, t]$ and $B[0, \ldots, t]$ where all entries of B are in $\{1, \infty\}$, we can compute the (min, +)-convolution of A and B in $\tilde{O}(t^{3/2})$ time.*

Furthermore, if we just want t' user-specified entries of the (min, +)-convolution, the time bound may be reduced to $\tilde{O}(t\sqrt{t'})$.

Proof.   By sorting and replacing elements by their ranks, we may assume the values of A are in $[t]$, and are distinct (without loss of generality). Divide the range $[t]$ into $\sqrt{t'}$ subintervals of length $t/\sqrt{t'}$. For each such subinterval I, define a Boolean array A'_I with $A'_I[j] = 1$ iff $A[j] \in I$, and define a Boolean array B' with $B'[j] = 1$ iff $B[j] \neq \infty$; compute the Boolean convolution between A'_I and B'; this requires $\sqrt{t'}$ FFTs and takes $\tilde{O}(t\sqrt{t'})$ time. Then for each index j for which we want to compute the output entry, we can now identify which subinterval contains the minimum answer (namely, the smallest subinterval I such that $(A'_I \circ B')[j]$ is true) in $O(\sqrt{t'})$ time, so we can do a brute-force search in $O(t/\sqrt{t'})$ time; the total time for t' output entries is $O(t' \cdot (\sqrt{t'} + t/\sqrt{t'})) = O(t\sqrt{t'})$. ◀

3 $\tilde{O}(t^{4/3})$ Algorithm

Previous algorithm.   Before presenting the new algorithm, we first give a sketch on the previous $\tilde{O}(t^{3/2})$-time algorithm by Bringmann and Kociumaka (as mentioned in [19]). Let ℓ_0 be a parameter to be chosen later. Let $H = \{v_i : v_i > \ell_0\}$ be the set of all *heavy* coin values, and let $L = \{v_i : v_i \leq \ell_0\}$ be the set of all *light* coin values. Because the coin values are distinct, $|L| \leq \ell_0$. To sum to any value $j \leq t$, we can use at most t/ℓ_0 heavy coins. We use Boolean convolution to compute the array $C_H^{(k)}$ from $C_H^{(k-1)}$ for each $k = 1, \ldots, \lfloor t/\ell_0 \rfloor$. The total time for these $\lfloor t/\ell_0 \rfloor$ convolutions is $\tilde{O}(t^2/\ell_0)$. We can thus obtain $D_H[j]$ by taking the minimum $k \leq t/\ell_0$ such that $C_H^{(k)}[j] > 0$. To finish, we use the classical dynamic programming algorithm to add the light coins. Namely, for each $j = 1, \ldots, t$, we set $D_V[j] = \min\{D_H[j], \min_{v_i \in L} D_V[j - v_i] + 1\}$. This step takes $O(\ell_0 t)$ time. The overall running time is

$$\tilde{O}\left(\frac{t^2}{\ell_0} + \ell_0 t\right).$$

To balance cost, we choose $\ell_0 = \sqrt{t}$ and obtain a time bound of $\tilde{O}(t^{3/2})$.

New algorithm.   To improve the running time, we describe a more efficient way to add the light coins, by using $(\min, +)$-convolution. As before, we first compute D_H for the heavy coins in $\tilde{O}(t^2/\ell_0)$ time. Initialize S to H.

Now, consider a fixed value $\ell \le \ell_0/2$, and consider the subset of light coins $L_\ell = \{v_i : v_i \in (\ell, 2\ell]\}$. In order to add L_ℓ to S, we need to compute $D_{S \cup L_\ell}$ from D_S. Naively, one could perform a single $(\min, +)$-convolution of D_S with D_{L_ℓ}, but this is expensive, and D_{L_ℓ} is not known yet (and is not binary). A better approach is to do *multiple* $(\min, +)$-convolutions by dividing the array into smaller blocks of size $O(\ell)$, as follows:

For each $i = 0, \dots, t/\ell$, we compute $D_{S \cup L_\ell}[\ell i, \dots, \ell(i+1)]$ by taking a $(\min, +)$-convolution D' of $D_{S \cup L_\ell}[\ell(i-2), \dots, \ell i]$ with a binary array $B[\ell, \dots, 2\ell]$ using Lemma 1, where $B[j] = 1$ if $j \in L_\ell$, and $B[j] = \infty$ otherwise. Then $D_{S \cup L_\ell}[\ell i, \dots, \ell(i+1)]$ is the entry-wise minimum of $D'[\ell i, \dots, \ell(i+1)]$ and $D_S[\ell i, \dots, \ell(i+1)]$. Each of the above $O(t/\ell)$ $(\min, +)$-convolutions is done to arrays of size $O(\ell)$ (after shifting indices). Thus, the total running time is $\tilde{O}((t/\ell) \cdot \ell^{3/2}) = \tilde{O}(\sqrt{\ell}t)$.

We repeat the above steps for all ℓ's that are powers of 2 and smaller than ℓ_0, until all coin values are added to S. This requires $O(\log \ell_0)$ rounds, and the total running time forms a geometric series bounded by $\tilde{O}(\sqrt{\ell_0}t)$. The overall running time is

$$\tilde{O}\left(\frac{t^2}{\ell_0} + \sqrt{\ell_0}t\right).$$

To balance cost, we choose $\ell_0 \approx t^{2/3}$ and obtain a time bound of $\tilde{O}(t^{4/3})$.

▶ **Theorem 2.** *The all-targets change-making problem can be solved in $\tilde{O}(t^{4/3})$ time.*

▶ Remark. If we choose $\ell_0 = u$ instead, the heavy coin case can be ignored and we obtain an $\tilde{O}(t\sqrt{u})$-time algorithm, which is faster for small u. We will give still faster algorithms for small u in the next section.

4 $O(u^2 \log u + t)$ Algorithm

We now explore more algorithms with running time sensitive to u.

Warm-up.   We first observe that there is a simple algorithm with $O(u^3 + t)$ running time. We use the following lemma, which is "folklore":[4]

▶ **Lemma 3.** *For any target value $j \ge u^2$, any optimal solution to the change-making problem must use the largest coin value u.*

Proof. Suppose that an optimal solution X for a target value j does not use the coin value u.

A simple argument shows that $j < u^3$: If X uses a coin value v_i at least u times, we can replace u copies of v_i with v_i copies of u, and the number of coins in X would decrease: a contradiction. Thus, each of the at most u coin values is used fewer than u times, and so the sum of X must be less than u^3.

---

[4] Bateni et al. [4, Lemma 7.2] gave a proof for the (more general) unbounded knapsack problem, using the pigeonhole principle, similar to what we give here (Eisenbrand and Weismantel [12] also proved a similar statement for higher-dimensional unbounded knapsack). But it was known much earlier: we personally learned of the pigeonhole proof for coin changing from comments by Bruce Merry in 2006 on a US Olympiad question (`https://contest.usaco.org/TESTDATA/DEC06.fewcoins.htm`), and the same pigeonhole proof for unbounded knapsack from a Chinese web post in 2016 (`https://www.zhihu.com/question/27547892/answer/133582594`).

We give a better argument showing $j < u^2$ by using the pigeonhole principle: Let $\langle x_1, \ldots, x_h \rangle$ be the sequence of coins used in X, with duplicates included, in an arbitrary order. Define the prefix sum $s_i = x_1 + \cdots + x_i$. Suppose $h \geq u$. By the pigeonhole principle, there must exist $0 \leq i < j \leq h$ with $s_i \equiv s_j \pmod{u}$. Then the subsequence $x_{i+1}, \ldots, x_j$ sums to a number divisible by u. We can replace this subsequence with some number of copies of u, and the number of coins in X would decrease (since u is the largest coin value): a contradiction. Thus $h < u$, and so the sum of X is less than u^2.   ◀

The above lemma ensures that it is sufficient to compute $D_V[j]$ for all $j < u^2$; by the naive dynamic programming algorithm, this step takes $O(nu^2) \leq O(u^3)$ time. Afterwards, for $j = u^2, \ldots, t$, we can simply set $D_V[j] = D_V[j - u] + 1$; this step takes $O(t)$ time. We thus get the time bound $O(u^3 + t)$.

If in the first part we instead use the $\tilde{O}(t\sqrt{u})$-time algorithm in the remark after Theorem 2 (with t replaced by u^2), then the first part takes $\tilde{O}(u^2\sqrt{u})$ time. The total time is then reduced to $O(u^{2.5} \operatorname{polylog} u + t)$. (This requires FFT, however.)

New algorithm.   To improve the running time further, we use number-theoretic results on the *Frobenius problem*, which has received much attention from mathematicians: given k positive integer coin values $v_1 > \cdots > v_k$ with $\gcd(v_1, \ldots, v_k) = 1$, what is the largest number that cannot be represented? For $k = 2$, classical results show that the number is exactly $v_1 v_2 - v_1 - v_2$. For $k \geq 3$, the problem becomes much more challenging, for which there are no closed-form formulas. In 1972, Erdős and Graham [13] proved an upper bound of $2 \lfloor \frac{v_1}{k} \rfloor v_2 - v_1$, which will be useful in our algorithmic application:

▶ **Lemma 4** (Erdős–Graham). *Given integers $v_1 > \ldots > v_k > 0$ ($k \geq 2$) with $\gcd(v_1, \ldots, v_k) = 1$, any integer greater than $2 \lfloor \frac{v_1}{k} \rfloor v_2 - v_1$ can be expressed as a nonnegative integer linear combination of $v_1, \ldots, v_k$.*

In terms of $u = \max_i v_i$, Erdős and Graham's bound is $O(u^2/k)$, which is known to be tight in the worst case, within a constant factor (see [10] for improvements on the constant factor). For constant k, the bound remains quadratic, as in the 2-coins case. In our algorithmic application, we will consider non-constant k – here, the k in the denominator will prove crucial.

First, let us restate the bound more generally without assuming $\gcd(v_1, \ldots, v_k) = 1$:

▶ **Corollary 5.** *Given integers $v_1 > \cdots > v_k > 0$ ($k \geq 2$) with $\gcd(v_1, \ldots, v_k) = d$, any integer that is greater than $2 \lfloor \frac{v_1}{dk} \rfloor v_2 - v_1$ and is divisible by d can be expressed as a nonnegative integer linear combination of $v_1, \ldots, v_k$.*

Proof.   Apply Lemma 4 to the numbers $v_1/d, \ldots, v_k/d$. The bound becomes $\left(2 \lfloor \frac{v_1/d}{k} \rfloor v_2/d - v_1/d\right) \cdot d$.   ◀

We use Corollary 5 to prove a more refined version of Lemma 3, which takes into account the k largest coin values instead of just the largest value:

▶ **Lemma 6.** *Let $v_1, \ldots, v_k \leq u$ be the k largest input coin values. For any target value $j \geq 2u^2/k$, any optimal solution to the change-making problem must use at least one coin from $\{v_1, \ldots, v_k\}$.*

Proof.   We may assume $k \geq 2$ (because of Lemma 3). Let $d = \gcd(v_1, \ldots, v_k)$. Suppose that an optimal solution X for a target value j does not use any coins from $\{v_1, \ldots, v_k\}$.

Consider the sequence of coins used in X, with duplicates included, in an arbitrary order. Divide the sequence into subsequences $X_1, \ldots, X_h$, each of which has sum in $(\frac{2u^2}{dk} - u, \frac{2u^2}{dk}]$, except that the last has sum at most $\frac{2u^2}{dk} - u$. Suppose $h > d$. Define s_i to be the sum of the concatenation of $X_1, \ldots, X_i$. By the pigeonhole principle, there exist $0 \le i < j < h$ with $s_i \equiv s_j \pmod{d}$. Then the subsequence formed by concatenating $X_{i+1}, \ldots, X_j$ sums to a number divisible by d and greater than $\frac{2u^2}{dk} - u$. By Corollary 5, we can replace this subsequence with coins from the set $\{v_1, \ldots, v_k\}$, and the number of coins in X would decrease (since $v_1, \ldots, v_k$ have larger values): a contradiction. Thus $h \le d$, and so the sum of X is less than $d \cdot \frac{2u^2}{dk} = 2u^2/k$. ◀

Thus, the optimal solution for target value j must use at least one coin value which is among the $\lceil 2u^2/j \rceil$ largest. This leads to the following extremely simple algorithm, which is just a small modification to the standard dynamic programming algorithm (no FFT required):

🟨 **Algorithm 1** All-targets change-making.

1: Sort $v_1, \ldots, v_n$ in decreasing order, and set $D_V[0] = 0$.
2: **for** $j = 1, \ldots, t$ **do**
3:     Set $D_V[j] = \min_{1 \le i \le \lceil 2u^2/j \rceil : \, v_i \le j} D_V[j - v_i] + 1$.

The total running time is bounded by a Harmonic series:

$$O\left(\sum_{j=1}^{t} \left(\frac{u^2}{j} + 1\right)\right) = O(u^2 \log u + t).$$

▶ **Theorem 7.** *The all-targets change-making problem can be solved in $O(u^2 \log u + t)$ time.*

As a corollary of the above algorithm, we can also obtain an algorithm with running time sensitive to σ, the total sum of the input coin values: Define the heavy coins H and light coins L as before, with respect to a parameter ℓ_0 to be chosen later. We first compute D_L for the light coins by the above algorithm in $\tilde{O}(\ell_0^2 + t)$ time. Then we add the heavy coins by dynamic programming: $D_V[j] = \min\{D_L[j], \min_{v_i \in H} D_V[j - v_i] + 1\}$. Since there are at most σ/ℓ_0 heavy coins, this step takes $O(\sigma/\ell_0 \cdot t)$ time. The overall running time is

$$\tilde{O}\left(\ell_0^2 + \frac{t\sigma}{\ell_0} + t\right).$$

To balance cost, we choose $\ell_0 = (t\sigma)^{1/3}$ and obtain the time bound $\tilde{O}((t\sigma)^{2/3} + t)$. (Again, no FFT is required.)

▶ **Corollary 8.** *The all-targets change-making problem can be solved in $\tilde{O}((t\sigma)^{2/3} + t)$ time.*

▶ **Remark.** The $O(t)$ term can be eliminated in Theorem 7 (and thus Corollary 8) if we are fine with an *implicit* representation of the output (i.e., a structure that allows us to return the answer for any given target in constant time), since by Lemma 3, we can first reduce the target j to below u^2 by using some number (i.e., $\max\{\lceil (j - u^2)/u \rceil, 0\}$) of copies of the largest coin value u.

5   All-Capacities Unbounded Knapsack

We note that the algorithm in the preceding section can be extended to solve the all-capacities version of the unbounded knapsack problem, defined as follows:

▶ **Problem 2** (ALL-CAPACITIES UNBOUNDED KNAPSACK). *Given n items where the i-th item has a positive integer weight w_i and a positive profit p_i, and given an integer t, for each $j = 0, \ldots, t$, find the maximum total profit of a multiset of items such that the total weight is at most j, i.e., find the maximum of $\sum_{i=1}^{n} m_i p_i$ subject to the constraint that $\sum_{i=1}^{n} m_i w_i \leq j$, where $m_i \in \mathbb{N}$.*

Like before, let $u = \max_{i=1}^{n} w_i$ and $\sigma = \sum_{i=1}^{n} w_i$. We may assume that the weights are distinct (since if there are two items with the same weight, we may remove the one with the smaller profit).

We use the following analog to Lemma 6:

▶ **Lemma 9.** *Suppose items $1, \ldots, k$ have the k largest profit-to-weight ratios. For any capacity value $j \geq 3u^2/k$, any optimal solution to the unbounded knapsack problem must use at least one item from $\{1, \ldots, k\}$.*

Proof. Similar to the proof of Lemma 6, since replacing a subsequence with items that have larger profit-to-weight ratios while maintaining the same total weight would increase the total profit. One difference in the unbounded knapsack problem is that the total weight in the optimal solution may not be exactly j. But it must be at least $j - u$ (otherwise, we could add one more item to get a better solution). When $j \geq 3u^2/k$, we have $j - u \geq 2u^2/k$. ◀

The same analysis shows correctness of the following very simple algorithm, which runs in $O(u^2 \log u + t)$ time:

🟧 **Algorithm 2** All-capacities unbounded knapsack.

1: Sort the items in decreasing order of p_i/w_i.
2: **for** $j = 0, \ldots, t$ **do**
3:     Set $D[j] = \max\{0, \ \max_{1 \leq i \leq \lceil 3u^2/j \rceil : \ w_i \leq j}(D[j - w_i] + p_i)\}$.

The $\tilde{O}((t\sigma)^{2/3} + t)$ algorithm can be extended as well.

▶ **Corollary 10.** *The all-capacities unbounded knapsack problem can be solved in $O(u^2 \log u + t)$ or $\tilde{O}((t\sigma)^{2/3} + t)$ time.*

▶ **Remark.** As before, the $O(t)$ term can be eliminated with an implicit representation of the output (since by an analog to Lemma 3, we can first reduce the capacity to below u^2 by using some number of copies of the item with the largest profit-to-weight ratio). In particular, for the single-capacity version, we obtain a very simple $O(u^2 \log u)$-time algorithm.

The algorithm works even when the profits are reals but the weights are integers. Alternatively, a variant of the algorithm works when the weights are reals but the profits are integers: the same time bound $O(u^2 \log u)$ holds but with $u = \max_{i=1}^{n} p_i$. Here, we recast the problem as minimizing $\sum_{i=1}^{n} m_i w_i$ subject to the constraint that $\sum_{i=1}^{n} m_i p_i \geq j$, and modify the algorithm appropriately (applying Erdős–Graham to the profits instead of the weights). From the implicitly represented output, we can determine the answer for any given capacity by predecessor search.

6 $\tilde{O}(u)$ Algorithm for Single-Target Change-Making

In this section, we present an $\tilde{O}(u)$-time algorithm for the single-target change-making problem. It is obtained by modifying the third algorithm of Chan and He [8], which originally ran in $O(t \log^2 t)$ time. They first solved the decision problem: deciding whether we can sum to t using at most m coins for a given value m. By adding 0 to the input set of coin values, "at most m" can be changed to "exactly m".

Their decision algorithm relies on the following partition lemma, which shows the multiset of coins S can be almost evenly partitioned simultaneously in terms of cardinality *and* the total value (see [8] for a short self-contained proof for the even case; the odd case is similar):

▶ **Lemma 11** (Partition Lemma). *Suppose S is a multiset with $|S| = m$ and $\sigma(S) = t$. If m is odd, then there exists a partition of S into three parts S_1, S_2 and a singleton $\{s_0\}$, such that $|S_1| = |S_2| = \frac{m-1}{2}$ and $\sigma(S_1), \sigma(S_2) \leq \frac{t}{2}$.*

If m is even, then there exists a partition of S into three parts S_1, S_2 and two singletons $\{s_0, s_1\}$, such that $|S_1| = |S_2| = \frac{m}{2} - 1$, and $\sigma(S_1), \sigma(S_2) \leq \frac{t}{2}$.

Notice that since the maximum coin value is u, we also have $\sigma(S_1), \sigma(S_2) \geq \frac{t}{2} - 2u$ (as we take out one or two coins).

The Partition Lemma suggests a simple recursive algorithm to compute $C_V^{(m)}[0, \ldots, t]$: we just take the first $t + 1$ entries of

$$
\begin{cases}
C_V^{(\frac{m-1}{2})}[0, \ldots, \frac{t}{2}] \circ C_V^{(\frac{m-1}{2})}[0, \ldots, \frac{t}{2}] \circ C_V^{(1)}[0, \ldots, t] & \text{if } m \text{ is odd,} \\
C_V^{(\frac{m}{2}-1)}[0, \ldots, \frac{t}{2}] \circ C_V^{(\frac{m}{2}-1)}[0, \ldots, \frac{t}{2}] \circ C_V^{(1)}[0, \ldots, t] \circ C_V^{(1)}[0, \ldots, t] & \text{if } m \text{ is even.}
\end{cases}
$$

That was essentially Chan and He's previous algorithm.

We describe a more efficient recursive algorithm to compute a smaller subarray $C_V^{(m)}[t - 4u, \ldots, t]$: we just take the relevant entries of

$$
\begin{cases}
C_V^{(\frac{m-1}{2})}[\frac{t-4u}{2} - 2u, \ldots, \frac{t}{2}] \circ C_V^{(\frac{m-1}{2})}[\frac{t-4u}{2} - 2u, \ldots, \frac{t}{2}] \circ C_V^{(1)}[0, \ldots, u] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{if } m \text{ is odd,} \\
C_V^{(\frac{m}{2}-1)}[\frac{t-4u}{2} - 2u, \ldots, \frac{t}{2}] \circ C_V^{(\frac{m}{2}-1)}[\frac{t-4u}{2} - 2u, \ldots, \frac{t}{2}] \circ C_V^{(1)}[0, \ldots, u] \circ C_V^{(1)}[0, \ldots, u] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{if } m \text{ is even.}
\end{cases}
$$

Each of the above Boolean convolutions is done to arrays of size $O(u)$ (after shifting indices), and thus takes $O(u \log u)$ time. The subarrays $C_V^{(\frac{m-1}{2})}[\frac{t-4u}{2} - 2u, \ldots, \frac{t}{2}] = C_V^{(\frac{m-1}{2})}[\frac{t}{2} - 4u, \ldots, \frac{t}{2}]$ and $C_V^{(\frac{m}{2}-1)}[\frac{t-4u}{2} - 2u, \ldots, \frac{t}{2}] = C_V^{(\frac{m}{2}-1)}[\frac{t}{2} - 4u, \ldots, \frac{t}{2}]$ can be computed by recursion. Thus, the running time satisfies the recurrence

$$T(m, t) = T\left(\left\lfloor \frac{m-1}{2} \right\rfloor, \frac{t}{2}\right) + O(u \log u),$$

which solves to $T(m, t) = O(u \log u \log t)$.

The decision problem can now be solved by inspecting the entry $C_V^{(m)}[t]$. We can find the optimal number of coins by binary search with $O(\log t)$ calls to the decision algorithm. By Lemma 3, we can first reduce t to below u^2 by repeatedly using the largest coin value. Therefore, the total running time is $O(u \log u \log^2 t) \leq O(u \log^3 u)$.

▶ **Theorem 12.** *The single-target change-making problem can be solved in $O(u \log^3 u)$ time.*

▶ Remark. The above algorithm shares some similarity with the $\tilde{O}(u^2)$ algorithm by Axiotis and Tzamos [2] for unbounded knapsack, which also involves logarithmically many convolutions on subarrays of size $O(u)$, except that they used $(\min, +)$-convolutions and a more naive parititioning that approximately halves t, but not m. In contrast, the above Partition Lemma is crucial to our faster algorithm for change-making.

7 $\tilde{O}(nu)$ Algorithm for Single-Capacity Unbounded Knapsack

In this section, we revisit the standard (single-capacity) version of the unbounded knapsack problem and present a new $\tilde{O}(nu)$-time algorithm (recall that $u = \max_i w_i$). This algorithm is simple (no FFT needed), and is based on the following combinatorial lemma, which is obtained by another pigeonhole argument:

▶ **Lemma 13.** *For the unbounded knapsack problem for a given capacity $j < t_0$, there exists an optimal solution that uses at most $\log t_0$ different types of items.*

In particular, in some optimal solution, there exists an item i that is used at least $\frac{j}{w_i \log t_0}$ times.

Proof. Consider an optimal solution that uses the minimum number of types of items. Let S be the set of items used in this solution, excluding multiplicities. If $|S| > \log t_0$, by the pigeonhole principle there must exist two different subsets S_1 and S_2 of S with the same total weight, multiplicities included (since there are $2^{|S|}$ subsets and t_0 integers between 0 and $t_0 - 1$). We can replace the items in $S_2 \setminus S_1$ with $S_1 \setminus S_2$, or vice versa (depending which of the two has smaller total value), and get a new solution that has the same total weight but has larger or equal total value. And if it has equal total value, the new solution uses a smaller number of types of items (since $S_2 \setminus S_1$ and $S_1 \setminus S_2$ are nonempty): a contradiction.

Thus, $|S| \leq \log t_0$. This also implies that some item contributes at least $\frac{j}{\log t_0}$ to the total weight.   ◀

Let $b := \lceil \log t_0 \rceil$. Let $D[j]$ be the maximum profit for the unbounded knapsack problem with capacity j. The above lemma implies the following recursive formula for all $j < t_0$:

$$D[j] = \max\left\{ 0, \ \max_{i=1}^{n}(D[j - w_i x_{ij}] + p_i x_{ij}) \right\} \qquad \text{where } x_{ij} := \left\lfloor \frac{j}{w_i b} \right\rfloor.$$

Note that $j - w_i x_{ij} \in \left[(1 - \frac{1}{b})j, \ (1 - \frac{1}{b})j + u\right]$.

The above formula allows us to compute the subarray $D[t, \ldots, t + bu]$ from the subarray $D\left[(1 - \frac{1}{b})t, \ldots, (1 - \frac{1}{b})(t + bu) + u\right] = D\left[(1 - \frac{1}{b})t, \ldots, (1 - \frac{1}{b})t + bu\right]$ in $O(bu \cdot n)$ time. The latter subarray can be computed recursively.

Let $T(t)$ denote the time for computing $D[t, \ldots, t + bu]$. We thus obtain the following recurrence:

$$T(t) = T((1 - \tfrac{1}{b})t) + O(bnu).$$

For the base case, we have $T(0) = O(bnu)$ by the standard dynamic programming algorithm (which computes $D[0, \ldots, j]$ in $O(nj)$ time). The number of levels of recursion is $O(b \log t)$. So, $T(t) = O(b^2 nu \log t) = O(nu \log^2 t_0 \log t)$. We can set $t_0 = (t + u)^{O(1)}$. As before, we can initially reduce the capacity t to below u^2 by repeatedly using the item with the largest profit-to-weight ratio. This yields the following result:

▶ **Theorem 14.** *The single-capacity unbounded knapsack problem can be solved in $O(nu \log^3 u)$ time.*

▶ **Remark.** There are alternative ways to exploit the above lemma to get $\tilde{O}(nu)$ algorithms (by computing $D[j]$ for a different choice of $\tilde{O}(bu)$ indices j), and the polylogarithmic factor is likely improvable.

8   Minimum Word Break

Bringmann, Grønlund, and Larsen [6] studied the decision version of the word break problem, and gave an algorithm with $\tilde{O}(nm^{1/3} + m)$ running time (with a matching conditional lower bound for combinatorial algorithms).

We consider the optimization version of the problem (with unit weight), defined as follows:

▶ **Problem 3** (MINIMUM WORD BREAK). *Given a string s with length n and a dictionary D with total length m, find the minimum number t^* such that s can be split into t^* words in D (duplicates are allowed).*

The single-target change-making problem can be viewed as a special case of this problem, by representing each coin with value v_i as a string with length v_i over a unary alphabet.

In this section, we briefly note that Bringmann et al.'s algorithm can be modified to solve the minimum word break problem without increasing the running time (ignoring polylogarithmic factors), by using our Lemma 1.

The modification. Since much of the solution proceeds as in Bringmann et al.'s paper [6], we will only describe the difference and assume the reader is already familiar with the previous paper. In particular, we will use the same notation.

Upon close inspection of their paper, we see that most parts of Bringmann et al.'s method require no (or straightforward) changes. Their "first algorithm" is no longer required, and the main change lies in their "second algorithm", specifically, the "query algorithm" in [6, Section 4.2 (arXiv version)]. Instead of computing $S + S_B$ using FFT, we now need to compute the $(\min, +)$-convolution between S and S_B, where $S_B[i] = 1$ if u_i is marked, and $S_B[i] = \infty$ otherwise. We are only interested in q_B entries in the output array, where $\sum_{B \in \mathcal{B}} q_B = O(q)$ and $|\mathcal{B}| \leq \frac{m}{q \cdot \lambda_q}$. (Note that $|\mathcal{B}| = 0$ if $\lambda_q > m/q$.) By using the output-sensitive bound from Lemma 1, we can perform the $(\min, +)$-convolution in $\tilde{O}(q\sqrt{q_B})$ time. By the Cauchy–Schwarz inequality, the sum of the cost over all $B \in \mathcal{B}$ is

$$\tilde{O}\left(\sum_{B \in \mathcal{B}} q\sqrt{q_B}\right) = \tilde{O}\left(q\sqrt{q|\mathcal{B}|}\right) = \tilde{O}\left(q\sqrt{\frac{m}{\lambda_q}}\right).$$

The other parts of the query algorithm requires $\tilde{O}(q \cdot \lambda_q)$ time. The total time is

$$\tilde{O}\left(q\lambda_q + q\sqrt{\frac{m}{\lambda_q}}\right).$$

To balance cost, we choose $\lambda_q = m^{1/3}$, and as a result, the query time in [6, Lemma 2 (arXiv version)] becomes $\tilde{O}(qm^{1/3})$, instead of $\tilde{O}(\min\{q^2, \sqrt{qm}\})$.

The final running time as analyzed in [6, page 10] was

$$\tilde{O}\left(\sum_{q=2^\ell} \frac{n}{q} \cdot \min\{q^2, \sqrt{qm}\}\right) = \tilde{O}(nm^{1/3}),$$

plus $\tilde{O}(n + m)$ for preprocessing. With the new query time bound, the sum changes to

$$\tilde{O}\left(\sum_{q=2^\ell} \frac{n}{q} \cdot qm^{1/3}\right) = \tilde{O}(nm^{1/3}),$$

which luckily gives the same result.

▶ **Theorem 15.** *The minimum word break problem can be solved in $\tilde{O}(nm^{1/3} + m)$ time.*

▶ Remark. The algorithm can actually solve an extension of the problem: compute the minimum number of breaks for every prefix of the input string. In particular, when the alphabet is unary, this implies an $\tilde{O}(t\sigma^{1/3} + \sigma)$-time algorithm for the all-targets change-making problem. However, this bound is not as good as those from Theorem 2 and Corollary 8 ($\tilde{O}(\min\{t^{4/3}, (t\sigma)^{2/3} + t\})$).

9 Concluding Remarks

Our change-making algorithms can be modified to compute not just the minimum number of coins but also a representation of the minimum multiset of coins for every target value. For the FFT-based algorithms, we need standard techniques for witness finding [1, 23] (which only increases the running time by polylogarithmic factors).

Although Erdős and Graham's $\Theta(u^2/k)$ bound on the Frobenius problem is asymptotically tight in the worst case (one bad coin set is $\{x, 2x, \ldots, (k-1)x, (k-1)x-1\}$ with $x = \lceil \frac{u}{k-1} \rceil$), the Frobenius number tends to be smaller for "many" k-tuples of coin values (it is usually subquadratic even for $k = 3$). This suggests that our $\tilde{O}(u^2 + t)$-time algorithm for all-targets coin changing might be improvable for many input sets of coins. However, obtaining an improvement in the worst case remains intriguingly open (this might require new results on the Frobenius problem – the interplay between combinatorial and algorithmic results seems worthy of further study).

References

1 Noga Alon, Zvi Galil, Oded Margalit, and Moni Naor. Witnesses for Boolean matrix multiplication and for shortest paths. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 417–426, 1992. `doi:10.1109/SFCS.1992.267748`.

2 Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster knapsack and graph algorithms. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 19:1–19:13, 2019. `doi:10.4230/LIPIcs.ICALP.2019.19`.

3 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 457–466, 2016.

4 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for knapsack via convolution and prediction. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1269–1282, 2018. `doi:10.1145/3188745.3188876`.

5 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1073–1084, 2017.

6 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 307–318, 2017. `arXiv:1611.00918`.

7 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010.

8 Timothy M. Chan and Qizheng He. On the change-making problem. In *Proceedings of the 4th ACM-SIAM Symposium on Simplicity in Algorithms (SOSA)*, pages 38–42, 2020.

9 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min, +)-convolution. *ACM Transactions on Algorithms*, 15(1):14:1–14:25, 2019. `doi:10.1145/3293465`.

10 Jacques Dixmier. Proof of a conjecture by Erdős and Graham concerning the problem of Frobenius. *Journal of Number Theory*, 34(2):198–209, 1990.

11 Ran Duan and Seth Pettie. Fast algorithms for (max,min)-matrix multiplication and bottleneck shortest paths. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–391, 2009.

12 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the Steinitz lemma. *ACM Transactions on Algorithms*, 16(1):5:1–5:14, 2020. `doi:10.1145/3340322`.

13   Paul Erdős and Ronald L Graham. On a linear diophantine problem of Frobenius. *Acta Arithmetica*, 21(1):399–408, 1972.

14   Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *Proceedings of the 10th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 43:1–43:17, 2019. `doi:10.4230/LIPIcs.ITCS.2019.43`.

15   Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In *Proceedings of the 2nd Symposium on Simplicity in Algorithms (SOSA)*, volume 69, pages 17:1–17:6, 2019.

16   Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1062–1072, 2017.

17   Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Transactions on Algorithms*, 15(3):1–20, 2019.

18   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.

19   Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 53:1–53:18, 2020.

20   George S. Lueker. Two NP-complete problems in nonnegative integer programming. Technical report, Princeton University. Department of Electrical Engineering, 1975.

21   Jiří Matoušek. Computing dominances in E^n. *Information Processing Letters*, 38(5):277–278, 1991. `doi:10.1016/0020-0190(91)90071-O`.

22   David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 33(1):1–14, 1999.

23   Raimund Seidel. On the all-pairs-shortest-path problem. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 745–749, 1992. `doi:10.1145/129712.129784`.

24   Arie Tamir. New pseudopolynomial complexity bounds for the bounded and other integer knapsack related problems. *Operations Research Letters*, 37(5):303–306, 2009. `doi:10.1016/j.orl.2009.05.003`.

25   Virginia Vassilevska, R. Ryan Williams, and Raphael Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory of Computing*, 5(1):173–189, 2009.

26   R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018. `doi:10.1137/15M1024524`.

27   J. W. Wright. The change-making problem. *Journal of the ACM*, 22(1):125–128, 1975.

28   Chao Xu. Word break with cost. `https://chaoxuprime.com/posts/2019-09-19-word-break-with-cost.html`, 2019.