On the Change-Making Problem

Timothy M. Chan* Qizheng He*

Abstract

Given a set of n non-negative integers representing a coin system, the *change-making problem* seeks the fewest number of coins that sum up to a given value t, where each type of coin can be used an unlimited number of times. This problem is a popular homework exercise in dynamic programming, where the textbook solution runs in $O(nt)$ time.

It is not hard to solve this problem in $O(t \operatorname{polylog} t)$ time by using convolution. In this paper, we present a simple deterministic $O(t \log t \log\log t)$ time algorithm, and later improve the running time to $O(t \log t)$ by randomization.

1 Introduction

In the *change-making* (or *coin changing*) problem, a cashier wants to give exactly t amount of money to the customer, choosing from a set of coins, where each type of coin has a positive integer value and has unlimited supply, and the cashier wants to minimize the number of coins used. More formally, given a set V of n positive integers (coin values) and a target number t, we want to select the fewest number of elements from V that sum to exactly t, where an element may be picked more than once. This problem is a popular exercise in algorithm textbooks, where the standard solution runs in $O(nt)$ time by dynamic programming [13]. (The naive greedy algorithm may fail to give optimal solutions for arbitrary sets of coin values.) The problem is known to be weakly NP-hard [11].

The change-making problem is closely related to many other well known problems with similar $O(nt)$-time textbook solutions, the most relevant of which include

- *subset sum*: select elements from V to sum to exactly t, but an element may be picked at most once, and a minimum solution is not required;
- *unbounded knapsack*: select elements from V to sum to at most t, and an element may be picked more than once, but we want to maximize a general linear objective function.

(There are a number of other interesting problems related to coin systems, such as the Frobenius problem, but these are not relevant to the present paper.)

Partially spurred by a renewed interest in fine-grained complexity, there have been a flurry of recent works that improve upon the standard $O(nt)$ solution for the subset sum problem, starting with Koiliaris and Xu's deterministic $O(\sqrt{n}t \operatorname{polylog} t)$-time algorithm [8, 9], followed by Bringmann's randomized $O(t \operatorname{polylog} t)$-time algorithm [3] and Jin and Wu's randomized $O(t \log t)$-time algorithm from SOSA last year [7]. All these improved algorithms use convolutions (i.e., FFT) or other algebraic techniques. There are also conditional lower bounds [1] to suggest that getting better than $t^{1-\varepsilon}$ running time might not be possible for subset sum (ignoring $n^{O(1)}$ factors). Of course, as a function of n, improving upon $2^{n/2}$ worst-case time has remained a longstanding open problem.

Despite all these recent works on subset sum, the change-making problem has not received as much attention, and we are not aware of any reported improvements. One paper [10] provided conditional hardness results for knapsack-related problems, including a stronger version of the change-making problem with a general linear objective function. Goebbels et al. [5] noted an $O(mt \log t)$-time algorithm for the decision problem of testing whether there is a solution with at most m coins, which is better for small m, but their solution uses convolution in a very naive way. The situation is somewhat ironic, considering that change-making seems not as difficult to solve via convolution, because duplicates are allowed (in previous works on solving subset sum via convolution [8, 3], much of the challenge is about how to avoid duplicates). Indeed, an $O(t \operatorname{polylog} t)$-time solution follows almost immediately by combining convolution with repeated squaring—we point out some of these easily rediscoverable solutions in Section 3.[1]

Following these straightforward solutions with extra logarithmic factors, we present a new simple deterministic algorithm in Section 4 that improves the running

*Department of Computer Science, University of Illinois at Urbana-Champaign, USA, {tmc,qizheng6}@illinois.edu. Work supported in part by NSF Grant CCF-1814026.

[1] For example, the first solution in Section 3 has appeared in an algorithms homework by Sariel Har-Peled; see the second page of `https://courses.engr.illinois.edu/cs473/fa2018/hw/hw_03.pdf`.

time to $O(t \log t \log\log t)$, and a further refinement using (Las Vegas) randomization in Section 5 that achieves $O(t \log t)$ expected running time.

We feel that the succession of improved solutions presented here is well suitable for teaching, considering the popularity and natural appeal of the change-making problem.

2 Preliminaries

The change-making problem can be formally defined as follows:

PROBLEM 2.1. (CHANGE MAKING (OPTIMIZATION VERSION)) *Given a set $V = \{v_1, \ldots, v_n\}$ of n nonnegative integers (coin values) and an integer t, find the smallest multiset S (duplicates allowed) contained in V such that S sums to exactly t, i.e., find the minimum m^* of $\sum_{i=1}^{n} m_i$ subject to the constraint that $\sum_{i=1}^{n} m_i v_i = t$, where $m_i \in \mathbb{N}$.*

For simplicity of presentation, we focus on finding the minimum number of coins m^*. It is possible to modify our algorithms to return the optimal multiset of coins without increasing the asymptotic running time; see the remark at the end of Section 4.

Without loss of generality, we assume that 0 is in V, since adding a coin value 0 does not affect the optimal solution. This assumption ensures monotonicity, which will be useful: if there is a solution using m coins, then there is a solution using m' coins for any $m' \geq m$. We may assume that $n \leq t$. Trivially, $m^* \leq t$ (unless $m^* = \infty$).

Some of our algorithms will solve the decision version of the problem as a stepping stone:

PROBLEM 2.2. (CHANGE MAKING (DECISION VERSION)) *Given a set $V = \{v_1, \ldots, v_n\}$ of n nonnegative integers (coin values), an integer t, and an integer m, decide whether $m^* \leq m$, i.e., whether there is a multiset S with m coins (counting duplicates) such that S sums to exactly t, i.e., whether there exist $m_i \in \mathbb{N}$ such that $\sum_{i=1}^{n} m_i v_i = t$ and $\sum_{i=1}^{n} m_i = m$.*

We use the standard word-RAM model with word length $w = \Theta(\log t)$ in this paper.

Convolution. Let $A[0, \ldots, t_1 - 1]$ be an array with t_1 elements, and let $B[0, \ldots, t_2 - 1]$ be an array with t_2 elements. The convolution of A and B is an array $A * B$ with $t_1 + t_2 - 1$ elements such that $(A * B)[i] = \sum_{j=0}^{t_1-1} A[j]B[i-j]$ for $0 \leq i \leq t_1 + t_2 - 2$ (where we assume out-of-range values are 0). Let $A *_t B$ denote the array containing the first $t+1$ elements of $A * B$.

LEMMA 2.3. *We can compute the convolution $A *_t B$ of two arrays A and B in deterministic $O(t \log t)$ time, by FFT.*

Boolean convolution. When the elements of A and B are Boolean values in $\{0, 1\}$, we can define the Boolean convolution of A and B (with respective size t_1 and t_2) to be an array $A \circ B$ with $t_1 + t_2 - 1$ elements such that $(A \circ B)[i] = \bigvee_{j=0}^{t_1-1} (A[j] \wedge B[i-j])$ for $0 \leq i \leq t_1 + t_2 - 2$ (where we assume out-of-range values are 0). Similarly let $A \circ_t B$ denote the array containing the first $t+1$ elements of $A \circ B$. It is easy to see $(A \circ B)[i] = 1$ iff $(A * B)[i] > 0$, so we can compute $A \circ_t B$ by Lemma 2.3 in deterministic $O(t \log t)$ time.

The following lemma states that speedup is possible with randomization, if we can tolerate error with probability p per entry. This improvement will be needed only in our final randomized algorithm in Section 5.

LEMMA 2.4. *Given two arrays A and B with size $O(t)$ and $0 < p < 1$, we can compute an array C in randomized $O(t \log \frac{1}{p})$ time such that for all $0 \leq i \leq t$:*

- *If $(A \circ_t B)[i] = 0$, then $C[i] = 0$.*
- *If $(A \circ_t B)[i] = 1$, then $\Pr[C[i] = 1] \geq 1 - p$.*

Proof. For $p = 1/2$, this was noted for example by Indyk [6]. Roughly, we use Freivald's technique [4] to reduce Boolean convolution to convolution over $GF(2)$, with constant error probability. The latter problem can be solved by packing $k = \Theta(\log t)$ bits and compute convolution of arrays with size $O(\frac{t}{k})$ over $GF(2^k)$ by FFT.

We can further reduce the error probability to any $p < 1$ by repeating $O(\log \frac{1}{p})$ times independently and taking the entry-wise or of the results. ∎

3 Basic Algorithms

We first present two basic algorithms for the change-making problem.

First solution. The first solution solves the decision version of the problem in $O(t \log^2 t)$ time, and the optimization version in $O(t \log^3 t)$ time. For any k, let $C_k[0, \ldots, t]$ denote the Boolean array where $C_k[j] = 1$ iff we can sum to exactly j using k coins. Given a value m, we can decide whether $m \geq m^*$ by computing $C_m[t]$. It is easy to compute C_1 in $O(t)$ time, and we can compute C_m recursively (by repeated squaring with respect to the operator $\circ_t$): if m is even, $C_m = C_{\frac{m}{2}} \circ_t C_{\frac{m}{2}}$, otherwise, $C_m = C_{m-1} \circ_t C_1$. The algorithm requires $O(\log t)$ FFTs, so the running time is $O(t \log^2 t)$ for the decision version.

To solve the optimization version, we can use a binary search for the optimal number of coins m^*, which requires $O(\log t)$ calls to the decision algorithm, so the total running time is $O(t \log^3 t)$.

Second solution. The second algorithm reimplements the binary search more carefully. In the following pseudocode, lines 1–3 first perform repeated squaring; this part requires $O(\log t)$ FFTs and takes $O(t \log^2 t)$ time.

In lines 4–9, at each iteration, we maintain a value m satisfying the invariant that $m^* \in (m, m + 2^i]$, and we also maintain its corresponding array C_m; as i is decreased to 0, we will have zoomed into the correct value of m^*. This part also requires $O(\log t)$ FFTs and takes $O(t \log^2 t)$ time. Thus, the total running time is $O(t \log^2 t)$.

1: Set $C_1[j] = 1$ iff there is a coin with value j.
2: **for** $i = 1, \ldots, \lfloor \log t \rfloor$ **do**
3: Set $C_{2^i} = C_{2^{i-1}} \circ_t C_{2^{i-1}}$.
4: Set $m = 0$.
5: Set $C_0[j] = 1$ iff $j = 0$.
6: **for** $i = \lfloor \log t \rfloor, \ldots, 0$ **do**
7: Set $C_{m+2^i} = C_m \circ_t C_{2^i}$.
8: **if** $C_{m+2^i}[t] = 0$ **then**
9: Set $m = m + 2^i$.
10: Return $m^* = m + 1$.

4 Improved Deterministic Algorithms

Third solution. Let $\sigma(S) = \sum_{s \in S} s$ denote the sum of a multiset S. Our next algorithm is based on the following intuition: for the optimal multiset of coins S with total value t, if we can partition S evenly into two parts S_1 and S_2 each with total value $\frac{t}{2}$, then we can reduce the target value, which would speed up the FFT computations.[2] This is not always possible, but the following partition lemma shows that if we take out an additional coin from S, a balanced partition exists.

LEMMA 4.1. (PARTITION LEMMA[3]) *Suppose S is a multiset with $|S| = m$ and $\sigma(S) = t$, where m is even. There exists a partition of S into three parts S_1, S_2 and a singleton $\{s_0\}$, such that $|S_1| = \frac{m}{2}$, $|S_2| = \frac{m}{2} - 1$, and $\sigma(S_1), \sigma(S_2) \le \frac{t}{2}$.*

Proof. Let $s_1, s_2, \ldots, s_m$ be the elements of S (in any order). Without loss of generality, assume that the first half has a smaller sum than the second half (otherwise, we can swap the two halves). Maintain a sliding window W containing exactly $\frac{m}{2}$ consecutive elements of $s_1, \ldots, s_m$. Initially, $W = \{s_1, \ldots, s_{\frac{m}{2}}\}$ and $\sigma(W) \le \frac{t}{2}$. At the end, $W = \{s_{\frac{m}{2}+1}, \ldots, s_m\}$ and $\sigma(W) \ge \frac{t}{2}$. Thus, at some moment in time, we must have $\sigma(W) \le \frac{t}{2}$ but $\sigma(W') \ge \frac{t}{2}$, where W' denotes the next window after W. We let $S_1 = W$, and s_0 be the unique element in $W' \setminus W$, and $S_2 = S \setminus (W \cup \{s_0\})$. Since $S_2 \subseteq S \setminus W'$, we have $\sigma(S_2) \le \frac{t}{2}$. ∎

We now use the Partition Lemma to obtain a simple algorithm for the decision problem.

LEMMA 4.2. *We can solve the decision version of the change-making problem (Problem 2.2) in $O(t \log t)$ time.*

Proof. Recall that $C_k[0, \ldots, t]$ denotes the Boolean array where $C_k[j] = 1$ iff we can sum to exactly j using k coins. More generally, we let $C_{k,s}[0, \ldots, s]$ denote the subarray containing the first $s + 1$ elements of C_k.

The Partition Lemma suggests the following algorithm: If m is even, we recursively compute $C_{m,t} = C_{\frac{m}{2}-1,\frac{t}{2}} \circ_t C_{\frac{m}{2}-1,\frac{t}{2}} \circ_t C_{1,t} \circ_t C_{1,t}$. (Note that when applying the Partition Lemma, we think of taking out one more singleton from S_1 to make its size equal to $\frac{m}{2} - 1$. Of course, the set S is not known to the algorithm, but the Lemma is used for showing correctness.) If m is odd, we recursively compute $C_{m,t} = C_{m-1,t} \circ_t C_{1,t}$.

Each convolution $\circ_t$ takes $O(t \log t)$ time by FFT. The array sizes in the recursion form a geometric series, so the total running time is

$$O\left(\sum_{i=0}^{\log t} \left(\frac{1}{2}\right)^i t \log t\right) = O(t \log t). \qquad \blacksquare$$

If we directly perform a binary search for m^*, we would need $O(\log t)$ calls to Lemma 4.2 and the running time for the optimization version is still $O(t \log^2 t)$. In the next solution, we will reduce the cost of the binary search, by using the idea from the second solution...

Fourth solution. For our next algorithm, we need a lopsided variant of the Partition Lemma that can split into two parts of arbitrary sizes. Because we only need an upper bound on the total value for one of the two parts, the proof becomes completely trivial.

LEMMA 4.3. (LOPSIDED PARTITION LEMMA) *Suppose S is a multiset with $|S| = m$ and $\sigma(S) = t$. For any integers $m_1, m_2 \ge 0$ with $m_1 + m_2 = m$, there exists a partition of S into two parts S_1 and S_2, such that $|S_1| = m_1$, $|S_2| = m_2$, and $\sigma(S_1) \le \frac{m_1 t}{m}$.*

Proof. Just let S_1 contain the m_1 smallest elements of S. ∎

[2] Bringmann [3, Section 5] adopted a similar strategy to solve the unbounded subset sum problem (just deciding whether there exists a multiset of S summing to t), but as the coin problem seeks to minimize the cardinality of S, we want a partition that divides both the sum and the cardinality evenly—this is less obvious.

We first compute a 2-approximation for m^*, or more precisely, a value 2^ℓ such that $m^* \in (2^\ell, 2^{\ell+1}]$. This can be done by a binary search over all $O(\log t)$ powers of 2, which requires $O(\log\log t)$ calls to the decision algorithm, taking $O(t \log t \log\log t)$ time by Lemma 4.2.

Now, we proceed as in the second solution and maintain a value m satisfying $m^* \in (m, m + 2^i]$, along with the corresponding array $C_{m,t}$, and decrease i at each iteration to zoom into m^*, as shown in the pseudocode below. Note that the algorithm in Lemma 4.2 not only can solve the decision problem for one target value t but can compute the entire array $C_{m,t}$, for a given m.

1: Set $m = 2^\ell$.
2: Compute $C_{m,t}$ by Lemma 4.2.
3: **for** $i = \ell, \ldots, 0$ **do**
4: Set $C_{m+2^i,t} = C_{m,t} \circ_t C_{2^i, \frac{t}{2^{\ell-i}}}$, where $C_{2^i,\frac{t}{2^{\ell-i}}}$ is computed by Lemma 4.2.
5: **if** $C_{m+2^i,t}[t] = 0$ **then**
6: Set $m = m + 2^i$.
7: Return $m^* = m + 1$.

In line 4, the decrease in the subscript from t to $\frac{t}{2^{\ell-i}}$ in the second term is the key to improving the running time, and is due to the Lopsided Partition Lemma, since $\frac{2^i t}{m+2^i} \le \frac{t}{2^{\ell-i}}$. However, as stated in the pseudocode, each FFT still involves $O(t)$ elements and takes $O(t \log t)$ time per iteration, and the total running time is still $O(t \log^2 t)$.

We observe that it suffices to maintain just the last $\frac{t}{2^{\ell-i-1}} + 1$ elements of $C_{m,t}$, i.e., $C_{m,t}[t - \frac{t}{2^{\ell-i-1}}, \ldots, t]$, in each iteration. To compute $C_{m+2^i,t}[t']$ for any $t' \in [t - \frac{t}{2^{\ell-i}}, \ldots, t]$, we only need the values of $C_{m,t}[t'']$ for $t'' \in [t' - \frac{t}{2^{\ell-i}}, t'] \subseteq [t - \frac{t}{2^{\ell-i-1}}, \ldots, t]$. The invariant is maintained when we decrement i for the next iteration. This way, the convolution in line 4 involves only $O(\frac{t}{2^{\ell-i}})$ elements and takes $O((\frac{t}{2^{\ell-i}}) \log t)$ time. The total time over all iterations is bounded by a geometric series:

$$O\left(\sum_{i=0}^{\ell} \left(\frac{t}{2^{\ell-i}}\right) \log t\right) \;=\; O(t \log t).$$

The overall running time is $O(t \log t \log\log t)$, as the initial step for computing the 2-approximation turns out to be the bottleneck.

5 Improved Randomized Algorithm

Final solution. To speed up the fourth solution, it suffices to compute the initial 2-approximation faster. We do so by first improving the running time of Lemma 4.2 in a randomized setting where error is allowed.

LEMMA 5.1. *We can solve the decision version of the change-making problem (Problem 2.2) in $O(t(\log\log t + \log\frac{1}{p}))$ time, with error probability $O(p)$.*

Proof. We modify the algorithm in Lemma 4.2. As before, if m is even, we recursively compute $C_{m,t} = C_{\frac{m}{2}-1,\frac{t}{2}} \circ_t C_{\frac{m}{2}-1,\frac{t}{2}} \circ_t C_{1,t} \circ_t C_{1,t}$. If m is odd, we recursively compute $C_{m,t} = C_{m-1,t} \circ_t C_{1,t}$.

We make one change: For the top $h_0 = \log\log t$ levels of recursion (where in a "level", t is reduced to $t/2$), we use Lemma 2.4 instead to perform the Boolean convolutions, with one-sided error probability $\frac{p}{\log t}$. Consider the final array entry $C_{m,t}[t]$. If it is true, it will be detected correctly if certain $O(2^{h_0})$ entries in the top h_0 levels are all computed correctly (there are no errors below these levels). By a union bound, the probability of making an error in computing $C_{m,t}[t]$ is thus at most $O(2^{h_0} \cdot \frac{p}{\log t}) = O(p)$. The running time for the top h_0 levels is bounded by

$$O\left(\sum_{i=1}^{h_0} \left(\frac{1}{2}\right)^i t \cdot \log\left(\frac{\log t}{p}\right)\right) \;=\; O(t(\log\log t + \log\tfrac{1}{p})).$$

The remaining levels of recursion are unchanged and has running time

$$O\left(\sum_{i=h_0}^{\log t} \left(\frac{1}{2}\right)^i t \log t\right) \;=\; O\left(\frac{1}{\log t} \cdot t \log t\right) = O(t).$$

∎

Now we can compute a 2-approximation of m^*, i.e., a value 2^ℓ such that $m^* \in (2^\ell, 2^{\ell+1}]$, by binary search over all $O(\log t)$ powers of 2, by making $O(\log\log t)$ calls to the decision algorithm in Lemma 5.1 with error probability $p = \frac{1}{\log t}$. The running time is $O(t(\log\log t)^2)$. By a union bound, the overall error probability is $O(\frac{\log\log t}{\log t}) = o(1)$. We can verify whether the computed value 2^ℓ correctly satisfies $m^* > 2^\ell$ and $m^* \le 2^{\ell+1}$ by two additional calls to the (deterministic) decision algorithm from Lemma 4.2, in $O(t \log t)$ time. In expectation we only need to repeat $O(1)$ times. The total expected running time is thus $O(t \log t)$.

After computing 2^ℓ, the rest of the algorithm is as in the fourth solution and takes $O(t \log t)$ time.

Remark on finding an optimal multiset. If we want to recover an optimal multiset S of coins, we can first find the minimum number m^* and then run the decision algorithm in Lemma 4.2 for m^*, which computes the array $C_{m^*,t}$ in $O(t \log t)$ time. Say m^* is even (the odd case is similar). Note that during its execution, the algorithm has already computed the intermediate arrays $C_{\frac{m^*}{2}-1,\frac{t}{2}}$, $C_{\frac{m^*}{2}-1,\frac{t}{2}} \circ_t C_{\frac{m^*}{2}-1,\frac{t}{2}}$, and

$C_{\frac{m^*}{2}-1,\frac{t}{2}} \circ_t C_{\frac{m^*}{2}-1,\frac{t}{2}} \circ_t C_{1,t}$. By scanning these arrays in $O(t)$ time, we can decompose t into a sum of t_1, t_2, s_1, s_2, where $s_1, s_2 \in V$, $t_1, t_2 \leq \frac{t}{2}$, and $C_{\frac{m^*}{2}-1,\frac{t}{2}}[t_1]$ and $C_{\frac{m^*}{2}-1,\frac{t}{2}}[t_2]$ are true. We recursively find multisets of $\frac{m^*}{2}-1$ coins summing to t_1 and t_2. The extra running time satisfies the recurrence $T(t) = 2T(\frac{t}{2}) + O(t)$, which solves to $O(t \log t)$.

This represents another advantage of our improved algorithms over the basic algorithms: if we were using the first two solutions from Section 3 instead, recovering an optimal multiset efficiently would be less straightforward and would need "witness finding" techniques (e.g., see [2, 12]) for convolutions, which require more extra logarithmic factors.

Acknowledgement. We thank Sariel Har-Peled and Chao Xu for helpful discussion; one of Sariel's homework problems sparked the present work.

References

[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 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 41–57, 2019.

[2] 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.

[3] 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.

[4] Rusins Freivalds. Probabilistic machines can use less running time. In *IFIP congress*, volume 839, page 842, 1977.

[5] Steffen Goebbels, Frank Gurski, Jochen Rethmann, and Eda Yilmaz. Change-making problems revisited: a parameterized point of view. *Journal of Combinatorial Optimization*, 34(4):1218–1236, 2017.

[6] Piotr Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 166–173, 1998.

[7] 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.

[8] 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.

[9] Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Transactions on Algorithms (TALG)*, 15(3):40, 2019.

[10] 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)*, volume 80, pages 21:1–21:15, 2017.

[11] George S Lueker. *Two NP-complete problems in nonnegative integer programming.* Princeton University. Department of Electrical Engineering, 1975.

[12] 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.

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