EXACT ANALYSIS OF EXACT CHANGE: THE K-PAYMENT PROBLEM

BOAZ PATT-SHAMIR[†], YIANNIS TSIOUNIS[‡], AND YAIR FRANKEL[§]

Abstract. We introduce the k-payment problem: given a total budget of N units, the problem is to represent this budget as a set of coins, so that any k exact payments of total value at most N can be made using k disjoint subsets of the coins. The goal is to minimize the number of coins for any given N and k, while allowing the actual payments to be made on-line, namely without the need to know all payment requests in advance. The problem is motivated by the electronic cash model, where each coin is a long bit sequence, and typical electronic wallets have only limited storage capacity. The k-payment problem has additional applications in other resource-sharing scenarios.

Our results include a complete characterization of the k-payment problem as follows. First, we prove a necessary and sufficient condition for a given set of coins to solve the problem. Using this characterization, we prove that the number of coins in any solution to the k-payment problem is at least $kH_{N/k}$, where H_n denotes the nth element in the harmonic series. This condition can also be used to efficiently determine k (the maximal number of exact payments) which a given set of coins allows in the worst case. Secondly, we give an algorithm which produces, for any N and k, a solution with minimal number of coins. In the case that all denominations are available, the algorithm finds a coin allocation with at most $(k+1)H_{N/(k+1)}$ coins. (Both upper and lower bounds are the best possible.) Finally, we show how to generalize the algorithm to the case where some of the denominations are not available.

Key words. k-payment problem, electronic cash, exact change, coin allocation, change making.

1. Introduction. Consider the following everyday scenario. You want to withdraw N units of money from your bank. The teller asks you "how would you like to have it?" Let us assume that you need to have "exact change," i.e., given any payment request $P \leq N$, you should be able to choose a subset of your "coins" whose sum is precisely P. Let us further assume that you would like to withdraw your N units with the least possible number of coins. In this case, your answer depends on your estimate of how many payments you are going to make. In the worst case, you may be making N payments of 1 unit each, forcing you to take N coins of denomination 1. On the other extreme, you may need to make only a single payment P. In this case, even if you don't know P in advance, $\log_2(N+1)$ coins are sometimes sufficient (as we explain later). In this article, we provide a complete analysis of the general question, which we call the *k-payment problem:* what is the smallest set of coins which enables one to satisfy any k exact payment requests of total value up to N.

Motivation. In any payment system, be it physical or electronic, some transactions require payments of exact amounts. Forcing shops to provide change to a customer, if she does not possess the exact change, simply shifts the problem from the customers to the shops. The number of coins is particularly important in electronic cash (see, e.g., [2, 3, 12]), because electronic coins are inherently long bit sequences and their handling is computationally intensive, while the typical "smart-card" used to store them has small memory space and computational power [7].

Another interesting application of the problem arises in the context of resource sharing. For concreteness, consider a communication link whose total bandwidth is N. When the link is shared by time-multiplexing, there is a fixed schedule which assigns the time-slots (say, cells in ATM lines) to the different connections. Typical

[†]Dept. of Electrical Engineering, Tel Aviv University. Email: boaz@eng.tau.ac.il.
[‡]InternetCash Corp. Email: yiannis@internetcash.com.
[§]CertCo. Email: yfrankel@cryptographers.com.

schedules have small time slots, since assigning big slots to small requests entails under-utilization. It is important to note, however, that there is an inherent fixed overhead associated with each time slot (e.g., the header of an ATM cell). It would be therefore desirable to have a multiplexing schedule (with time slots of various sizes) which can accommodate any set of requests with the least number of slots. Similarly to the withdrawal scenario, an improvement upon the trivial N unit-slot solution can be achieved, if we know how many connections might be running in parallel. When we know a bound k on this number, and if we restrict ourselves to long-lived connections, the problem of designing a schedule naturally reduces to the k-payment problem.

The k-payment problem: Definition. Formally, the problem is as follows. There are two parameters, the *budget*, denoted N, and the *number of payments*, denoted k. The problem is to find, for each $i \geq 1$, the *number of i-coins* (also called *coins of denomination i*), denoted c_i, such that the following two requirements are satisfied.

Budget compliance: $N = \sum_{i=1}^{\infty} ic_i$, and

k-partition: For any sequence of k *payment requests*, denoted $P_1, P_2, \ldots, P_k$, with $\sum_{j=1}^{k} P_j \leq N$, there exists a way to exactly satisfy these payments using the coins. That is, there exist non-negative integers p_{ij} (where p_{ij} represents the number of i-coins used in the j-th payment), such that

- $\sum_i ip_{ij} = P_j$ for each $1 \leq j \leq k$, and
- $\sum_j p_{ij} \leq c_i$ for each $i \geq 1$.

The problem can thus be broken into two parts as follows. The *coin allocation* problem is to partition N into coins given N and k, i.e., determining the c_i's. The *coin dispensing* problem is how to actually make a payment, i.e., given the c_i's and a P_j as input, produce the p_{ij} values.

There are a few possible variants of the k-payment problem. First, in many systems, not all denominations are available (for example, we do not know of any system with 3-coins). Even in the electronic cash realm, denominations may be an expensive resource. (This is because each denomination requires a distinct pair of secret/public keys of the central authority; see, e.g., [1, 8].) Thus an interesting variant of the allocation problem is the *restricted denominations* version, where the set of possible solutions is restricted to only those in which $c_i = 0$ if $i \notin \mathcal{D}$, for a given *allowed denomination set* $\mathcal{D}$.

Also, one may consider the *on-line* coin dispensing problem, where the algorithm is required to dispense coins after each payment request, without knowledge of future requests, or the *off-line* version, where the value of all k payments is assumed to be known before the first coin is dispensed.

Results. It turns out that the coin dispensing problem is easy: the greedy strategy works (in the worst-case sense), even in the on-line setting. Henceforth we consider only the on-line problem. Most of our results concern the coin allocation problem, and we sometimes refer to this part of the problem as the k-payment problem. Our basic result is a simple necessary and sufficient condition for a sequence $c_1, c_2, \ldots$ of coins to solve the allocation problem (Theorem 3.1). Using this characterization, we prove (in Theorem 3.6) a lower bound of $kH_{N/k} \approx k \ln(N/k)$ on the number of coins in any solution for all N and k, where H_n denotes the nth element of the harmonic series. The lower bound is the best possible in the sense that it is met with equality for infinitely many N's and k's (Theorem 3.8). The characterization can be used to efficiently determine the maximal number k for which a given collection of coins solves the k payment problem (Corollary 3.9). Our next major result is an efficient algorithm which finds a solution for *any* N and k using the least possible number

of coins. We first deal with the case where all denominations are allowed (Theorem 4.1). In this case the number of coins is never more than $(k+1)H_{N/(k+1)}$ (Theorem 4.7). Similarly to our lower bound, the upper bound is the best possible in general (Theorem 4.8). Finally, using the same ideas in a slightly more refined way, we extend the algorithm to the general case of restricted denomination set (Theorem 5.1).

Related work. To the best of our knowledge, the current work is the first to formulate the general k-payment problem, and hence the first to analyze it. One related classic combinatorial problem is *k-partition*, where the question is in how many ways can a natural number N be represented as a sum of k positive integers. This problem is less structured than the k-payment problem, and can be used to derive lower bounds (however, these bounds are suboptimal: see §2). The *postage-stamp problem* is also closely related: cast in our terms, the postage-stamp problem is to find a set of denominations which will allow to pay any request of value $1, 2, 3, \ldots, N$ using at most h coins, so that N is maximized. The postage-stamp problem can be viewed as an inverse of the 1-payment problem: there is one payment to make, the number of coins is given, and the goal is to find a denomination set of a given size which will maximize the budget. We remark that the postage-stamp problem is considered a difficult problem even for a very small number of denominations. See, e.g., [10, 11, 4]. Another related question is *change making* [6], which is the problem of how to represent a given budget with the least number of coins from a given allowed denomination set. General change-making is (weakly) NP-hard. Kozen and Zaks [5], Verma and Xu [14], and Pearson [9], study the question of which denominations sets allow one to use the greedy strategy for optimal change making.

Organization. In §2 we introduce notation, give some preliminary observations and briefly discuss a few suboptimal results. In §3 we prove a characterization of the k-payment problem and a lower bound on the number of coins in any solution. In §4 we present and analyze an optimal algorithm for the unrestricted denomination case. In §5 we extend the algorithm for the case of restricted denominations. We conclude in §6 with a few open problems.

2. Notation and simple results. In this section we develop some intuition for the k-payment problem by presenting a few simple upper and lower bounds on the number of coins required. The notation we shall use throughout this article is summarized in Figure 1. The solution $\mathcal{S}$ to which the c_i, T_i and m symbols refer should be clear from the context.

For the remainder of this article, fix N and k to be arbitrary given positive integers. Note that we may assume without loss of generality that $k \leq N$, since payment requests of value 0 can be ignored.

Let us now do some rough analysis of the k-payment problem. As already mentioned above, the case $k = N$ is trivial to solve: take $c_1 = k$ and $c_i = 0$ for $i \neq 1$. Clearly, no better solution is possible since $c_1 < k$ would not satisfy k payments of value 1 each. The case of $k = 1$ is also quite simple, at least when $N = 2^g - 1$ for an integer $g \geq 1$: we can solve it with $g = \log_2(N+1)$ coins of denominations $1, 2, 4, \ldots, 2^{g-1}$. Given any request P, we can satisfy it by using the coins which correspond to the ones in the binary representation of P.

However, it is not immediately clear how can one generalize this solution to arbitrary N and k. Consider an arbitrary N: the usual technique of "rounding up" to the next power of 2 does not seem appropriate in the k-payment problem: can we ask the teller of the bank to round up the amount we withdraw just because it is more convenient for us? But let us ignore this point for the moment, and consider the

Parameters of problem specification:

- N: the total budget.
- k: the number of payments.
- $\mathcal{D}$: the set of allowed denominations. In the unrestricted case, $\mathcal{D} = \mathbf{N}$.

Quantities related to solution specification $\mathcal{S}$:

- c_i ($i \geq 0$): the number of coins of denomination i (a.k.a. i-coins) in $\mathcal{S}$. By convention, $c_0 = 0$.
- m: the largest denomination of a coin in $\mathcal{S}$. Formally, $m = \max\{i \mid c_i > 0\}$.
- T_i ($i \geq 0$): the budget allocated in $\mathcal{S}$ using coins of denomination i or less. Formally, $T_i = \sum_{j=1}^{i} jc_j$.
- It is required that $T_m = N$.

Quantities related to making a payment:

- c'_i, T'_i, m': refer to the respective quantities after a payment has been made.

Standard quantities:

- $H_n = \sum_{i=1}^{n} \frac{1}{i}$. By convention, $H_0 = 0$.

FIG. 1. *Glossary of notation.*

problem of general k. If we were allowed to make the dubious assumption that we may enlarge N, then one simple solution would be to duplicate the solution for 1-payment k times, and let each payment use its dedicated set of coins. Specifically, this means that we allocate k 1-coins, k 2-coins, k 4-coins and so on, up to k $2^{\lceil \log_2(N+1) \rceil - 1}$-coins. The result is approximately $k \log_2 N$ coins, and the guarantee we have based on this simplistic construction is that we can pay k payments, but for all we know *each* of these payments must be of value at most N. However, the total budget allocated in this solution is in fact kN, and thus it does not seem to solve the k-payment problem as stated, where the only limit on the value of payments is placed on their *sum*, rather than on individual values.

As an aside, we remark that one corollary of our work (specifically, Theorem 3.1) is that if coin dispensing is done using the greedy strategy (see below), then the above "binary" construction for coin allocation indeed solves the general k-payment problem. More precisely, assume that $(N/k) + 1$ is a power of 2. Then the coin allocation algorithm allocates k coins of each denomination $1, 2, 4, \ldots, 2^{\log_2((N/k)+1)-1}$. Clearly, the number of coins is $k \log_2((N/k) + 1)$ and their sum is N. Coin dispensing is made *greedily:* at each point, the largest possible coin is used. (A formal description of the greedy dispensing algorithm is presented in Figure 2.) However, one should note that, perhaps surprisingly, our results also indicate that the binary algorithm is *not* the right generalization for $k > 1$: the best algorithm (described in §4) yields a factor of about $(1 - \ln 2)$ improvement, i.e., roughly 30% fewer coins.

Let us now re-consider the question of a general N. Once we have an algorithm for infinitely many values of k and N, generalizing to arbitrary N and k is easy: find a solution for N' and k', with $N' \geq N$ and $k' \geq k+1$, then dispense a payment of value $N' - N$. The remaining set of coins is a coin allocation of total budget N, and it can be used for k additional payments since the original set solved the $(k + 1)$-payment problem.

We close this section with a simple lower bound on the number of coins required in any solution to the k-payment problem. The bound is based on a counting argument, and we only sketch it here. For $k = 1$, the number of distinct possible payment requests is $N + 1$ (0 is allowed). Observe that the algorithm must dispense a different

```
GreedyDispense(P)
1  while P > 0
2      do i ← max {l | l ≤ P, c_l > 0}        ▷ i is highest possible denomination
3         j ← min(⌊P/i⌋, c_i)                 ▷ use as many i-coins as possible
4         dispense j i-coins
5         c_i ← c_i − j
6         P ← P − j · i
```

FIG. 2. *The greedy algorithm for coin dispensing. P is the amount to be paid.*

set of coins in response to each request. It follows that the number of coins in any solution must be at least $\log_2(N+1)$. This argument can be extended to a general k, using the observation that the number of distinct responses of the algorithm (disregarding order), is at least $p_k(N)$, the number of ways to represent N as a sum of k positive integers. Using standard bounds for partitions (see, e.g., [13]), and since the number of responses is exponential in the number of coins, one can conclude that the number of coins is $\Omega(k \log(N/k^2))$.

3. Problem characterization. In this section we first prove a simple condition to be necessary and sufficient for a set of coins to correctly solve the k-payment problem. Using this result, we obtain a sharp lower bound on the number of coins in any solution to the k-payment problem. Finally, we outline an efficient algorithm which, given a set of coins $\mathcal{S}$, determines the maximal k for which $\mathcal{S}$ is a solution of the k-payment problem. Please refer to Figure 1 for notation.

3.1. A necessary and sufficient condition. THEOREM 3.1. *Let $\mathcal{S}$ be a solution as in Figure 1. Then $\mathcal{S}$ solves the k-payment problem if and only if $T_i \geq ki$ for all $0 \leq i < m$.* The theorem is proven by a series of lemmas below. The necessity proof is not hard: the intuition is that the "hardest" cases are when all payment requests are equal. The more interesting part is the sufficiency proof. We start by proving an upper bound on m, the largest denomination in a solution.

LEMMA 3.2. *If $\mathcal{S}$ solves the k-payment problem, then $m \leq \lceil N/k \rceil$.*

Proof. By contradiction. Suppose $m > \lceil N/k \rceil$, and consider k payments of values $\lfloor N/k \rfloor$ and $\lceil N/k \rceil$ such that their total sum is N. Clearly, none of these payments can use m-coins, and since $c_m \geq 1$ by definition, the total budget available for these payments is at most $N - m$, contradiction. □

The following lemma is slightly stronger than the condition in Theorem 3.1. We use this version in the proof of Theorem 3.6.

LEMMA 3.3. *If $\mathcal{S}$ solves the k-payment problem, then $T_i \geq ki$ for all $0 \leq i \leq \lfloor N/k \rfloor$.*

Proof. Let $i \leq \lfloor N/k \rfloor$. Then $ki \leq N$. Consider k payments of value i each: each such payment can be done only with coins of denomination at most i, hence $T_i \geq ki$. □

We now turn to sufficiency. We start by showing that if the condition of Theorem 3.1 holds for $k = 1$, then any single payment of value up to N can be satisfied by $\mathcal{S}$ under the greedy algorithm.

LEMMA 3.4. *Let P be a payment request, and suppose that in $\mathcal{S}$ we have that for some j,*

1. *$P \leq T_j$, and*
2. *$T_i \geq i$ for all $0 \leq i < j$.*

Then P can be satisfied by the greedy algorithm using only coins of denomination j or less.

Proof. We prove, by induction on j, that the claim holds for j and any P. The base case is $j = 1$: then by (1) $T_1 \geq P$ and hence there are at least P 1-coins, which can be used to pay any amount up to their total sum. For the inductive step, assume that the claim holds for j and all P, and consider $j + 1$. Let a be the number of $(j + 1)$-coins dispensed by the greedy algorithm, namely,

$$a = \min\left(\left\lfloor \frac{P}{j+1} \right\rfloor, c_{j+1}\right) \ .$$

Let R denote the remainder of the payment after the algorithm dispenses the $(j+1)$-coins, i.e., $R = P - a(j + 1)$. Note that (2) trivially holds after dispensing the $(j + 1)$-coins; we need to show that (1) holds as well. We consider two cases. If $a = c_{j+1}$, then using (1) we get

$$\begin{aligned} T_j &= T_{j+1} - c_{j+1}(j+1) \\ &= T_{j+1} - a(j+1) \\ &\geq P - a(j+1) \\ &= R \ , \end{aligned}$$

and we are done for this case.

If $a < c_{j+1}$, then since the algorithm is greedy, it must be the case that $R < j+1$. On the other hand, by (2) we have that $T_j \geq j$, and hence $T_j \geq R$ and we are done in this case too. □

The following lemma is the key invariant preserved by the greedy algorithm. It is interesting to note that while the algorithm proceeds from larger coins to smaller ones, the inductive proof goes in the opposite direction. Recall that "primed" quantities refer to the value after a payment is done.

LEMMA 3.5. *If $T_i \geq ki$ holds for all $0 \leq i < m$, then after the greedy algorithm dispenses any amount up to T_m, $T'_i \geq (k-1)i$ holds for all $0 \leq i < m'$.*

Proof. By induction on i. For $i = 0$ the claim is trivial. Assume that the claim holds for all $l < i$, and consider i. Let $S_i = T_i - T'_i$ (S_i is the amount dispensed using coins of denomination at most i).

Define $j = \min\left\{j \mid j > i, c'_j > 0\right\}$, namely j is the smallest remaining denomination which is larger than i. Note that j is well defined since $i < m'$, namely i is not the largest remaining coin. Next, note that since all the coins of denomination $i + 1, i + 2, \ldots, j - 1$ (whose sum is $T_{j-1} - T_i$) were used by the algorithm, we have that

$$S_{j-1} = T_{j-1} - T_i + S_i \tag{1}$$

Now, observe that since the algorithm is greedy, and since at least one j-coin was not used by the algorithm, it must be the case that the total amount dispensed using coins of denomination smaller than j is less than j, i.e., $S_{j-1} \leq j - 1$. Using Eq. (1), we get that $T_i \geq T_{j-1} - j + 1 + S_i$. Finally, using the assumption applied to $j - 1$, we obtain

$$\begin{aligned} T'_i &= T_i - S_i \\ &\geq T_{j-1} - j + 1 \\ &\geq (j-1)k - j + 1 \end{aligned}$$

$$= (j-1)(k-1)$$
$$\geq i(k-1)\ . \qquad \square$$

We now complete the proof of the characterization.

Proof of Theorem 3.1: The necessity of the condition follows directly from Lemmas 3.2 and 3.3. For the sufficiency, assume that $T_i \geq ik$ for all $0 \leq i < m$, and consider a sequence of up to k requests of total value at most N. After the l-th request is served by the greedy algorithm, we have, by inductive application of Lemma 3.5, that $T'_i \geq (k-l)i$ for all $0 \leq i < m$. Moreover, by Lemma 3.4, any amount up to the total remainder can be paid from $\mathcal{S}$, so long as $k - l > 0$, which completes the proof. $\square$

3.2. A lower bound on the number of coins. Using Theorem 3.1, we derive a lower bound on the number of coins in any solution to the k-payment problem.

THEOREM 3.6. *The number of coins in any solution to the k-payment problem is at least $kH_{\lfloor N/k \rfloor} \approx k \ln \frac{N}{k}$.* We first prove a little lemma we use again in Theorem 4.7.

LEMMA 3.7. *For any number j,* $\displaystyle\sum_{i=1}^{j} c_i = \sum_{i=1}^{j-1} \frac{T_i}{i(i+1)} + \frac{T_j}{j}$.

Proof.

$$\begin{aligned}\sum_{i=1}^{j} c_i &= \sum_{i=1}^{j} \frac{T_i - T_{i-1}}{i} \\ &= \sum_{i=1}^{j-1} \left(\frac{T_i}{i} - \frac{T_i}{i+1} \right) + \frac{T_j}{j} \\ &= \sum_{i=1}^{j-1} \frac{T_i}{i(i+1)} + \frac{T_j}{j}\ . \qquad \square\end{aligned}$$

Proof of Theorem 3.6: By Lemma 3.7 and Lemma 3.3:

$$\begin{aligned}\sum_{i=0}^{\lfloor N/k \rfloor} c_i &= \sum_{i=1}^{\lfloor N/k \rfloor - 1} \frac{T_i}{i(i+1)} + \frac{T_{\lfloor N/k \rfloor}}{\lfloor N/k \rfloor} \\ &\geq \sum_{i=1}^{\lfloor N/k \rfloor - 1} \frac{ki}{i(i+1)} + k \\ &= k \sum_{i=1}^{\lfloor N/k \rfloor - 1} \frac{1}{i+1} + k \\ &= k(H_{\lfloor N/k \rfloor} - 1) + k \\ &= kH_{\lfloor N/k \rfloor}\ . \qquad \square\end{aligned}$$

The lower bound of Theorem 3.6 is the best possible in general, as shown in the following theorem.

THEOREM 3.8. *For any natural numbers n_1, n_2, there are infinitely many $N > n_1, k > n_2$ such that there exists a solution for the k-payment problem with budget N with exactly $kH_{\lfloor N/k \rfloor}$ coins.*

Proof. Choose a natural number m such that $m! > n_2$ and $m \cdot m! > n_1$, and set $k = m!$ and $N = km$. In this case the solution with $c_i = k/i$ for $1 \leq i \leq N/k$ solves

```
ALLOC(N, k)
1  c_i ← 0 for all i                                          ▷ initialize
2  t ← 0
3  i ← 0
4  while N − t > i                  ▷ consider denominations in increasing order
5      do i ← i + 1
6         if t < ki
7           then c_i ← min(⌈(ki − t)/i⌉, ⌊(N − t)/i⌋)        ▷ add i-coins but don't overflow
8                t ← t + ic_i
9  if N > t                                                   ▷ add remainder
10   then c_{N−t} ← c_{N−t} + 1
```

FIG. 3. *Algorithm for optimal solution of the k-payment problem with unrestricted denominations.*

the k-payment problem by Theorem 3.1, and its total number of coins is precisely $kH_{\lfloor N/k \rfloor}$. □

3.3. Determining k for a given set. Consider the "inverse problem:" we are given a set of coins, with c_i coins of denomination i for each i, and the question is how many payments can we make using these coins, i.e., find k. Note that k is well defined: we say that k is 0 if there is a payment request which cannot be satisfied by the set, and k is never more than the total budget. Theorem 3.1 can be directly applied to answer such a question efficiently, as implied by the following simple corollary.

COROLLARY 3.9. *Let $\mathcal{S}$ be a given coin allocation. Then $\mathcal{S}$ solves the k-payment problem if and only if $k \leq \min\{\lfloor T_i/i \rfloor \mid 1 \leq i < m\}$.*

4. An optimal algorithm for unrestricted denominations. We now present a coin allocation algorithm which finds a minimal solution to the k-payment for arbitrary N and k, assuming that all denominations are available. We first prove the optimality of the algorithm, and then give an upper bound on the number of coins it allocates. In §5 we generalize the algorithm to handle a restricted set of denominations.

The algorithm. Given arbitrary integers $N \geq k > 0$, the algorithm presented in Figure 3 finds an optimal coin allocation. Intuitively, the algorithm works by scanning all possible denominations $i = 1, 2, 3, \ldots$ and adding, for each i, the least number of i-coins which suffices to make $T_i \geq ki$. The number of coins c_i is thus an approximation of the i-th harmonic element multiplied by k. When the budget is exhausted, the remainder is added simply as a single additional coin.

4.1. Optimality of ALLOC.

THEOREM 4.1. *Let $\mathcal{S}$ denote the solution produced by* ALLOC. *Then $\mathcal{S}$ solves the k-payment problem using the least possible number of coins.*

Proof. Follows from Lemma 4.3 and Lemma 4.5 below. □

The important properties of the algorithm are stated in the following loop invariant.

LEMMA 4.2. *Whenever* ALLOC *executes line 4, the following assertions hold.*

(i) $t = \sum_{j=1}^{i} jc_j \leq N$.
(ii) *if* $N - t > i$ *then* $t \geq ik$.
(iii) *if* $i > 0$ *then* $t < i(k+1)$.

Proof. Line 4 can be reached after executing lines 1–3 or after executing the loop of lines 5–8. In the former case, we have $t = i = 0$ and the lemma holds trivially.

Suppose now that line 4 is reached after an iteration of the loop. We denote by t_0 the value of the variable t *before* the last execution of the loop. If lines 7 and 8 are not executed, then (i) holds trivially. If they are executed, then we have that $t = t_0 + ic_i$, and $t \leq \left\lfloor \frac{N-t_0}{i} \right\rfloor \cdot i + t_0 \leq N$, and hence (i) holds after the loop is executed. Also note that $t \leq t_0 + \left\lceil \frac{ki-t_0}{i} \right\rceil \cdot i < t_0 + (ki - t_0 + i) = i(k+1)$, and therefore (iii) holds true after the execution of the loop.

Finally, we prove that (ii) holds after executing the loop. If lines 7–8 are not executed, (ii) holds trivially. Suppose that lines 7–8 are executed, and that $i < N - t$. In this case, by line 8, $i < N - t_0 - ic_i$, which means that

$$c_i \;<\; \frac{N-t_0}{i} - 1 \;<\; \left\lfloor \frac{N-t_0}{i} \right\rfloor .$$

Therefore, by line 7, $c_i = \left\lceil \frac{ki-t}{i} \right\rceil$, and hence $t = t_0 + \left\lceil \frac{ki-t_0}{i} \right\rceil \cdot i \geq t_0 + ki - t_0 = ki$, and we are done. □

Using Lemma 4.2 and Theorem 3.1, correctness is easily proven. We introduce some notation to facilitate separate handling of the remainder:

- c_i^*, for all i, is the value of the c_i variable when line 9 is executed.
- $T_i^* = \sum_{j=1}^{i} j c_j^*$.

LEMMA 4.3. *$\mathcal{S}$ solves the k-payment problem.*

Proof. By (i) of Lemma 4.2, in conjunction with lines 9–10 of the code, we have that upon completion of the algorithm, $\sum_i ic_i = N$. By (ii) of Lemma 4.2, and since (by lines 4 and 10) the largest denomination is $m \leq N - T_m^*$, we have that $T_i \geq T_i^* \geq ki$ for all denominations $i < m$, and therefore, by Theorem 3.1, $\mathcal{S}$ solves the k-payment problem. □

Proving optimality takes more work. We first deal with the the coins allocated before line 9 is reached, and then show that even if the remainder was non-zero (and an additional coin was allocated in line 10), the solution $\mathcal{S}$ produced by ALLOC is still optimal. To this end, we fix an arbitrary solution $\mathcal{T}$ for the k-payment problem, and use the following additional notation:

- d_i is the number of i-coins in $\mathcal{T}$.
- U_i is the budget allocated in $\mathcal{T}$ using coins of denomination i or less: $U_i = \sum_{j=1}^{i} id_i$.
- n is the largest denomination of a coin in $\mathcal{T}$: Formally, $n = \max\{i \mid d_i > 0\}$.
- $\Delta_0 = 0$ and $\Delta_i = \Delta_{i-1} + d_i - c_i^*$ for $i > 0$.

Note that by the definitions, the following holds for all $i \geq 0$:

$$\Delta_i = \sum_{j=1}^{i} d_j - \sum_{j=1}^{i} c_j^* \tag{2}$$

$$d_i = c_i^* + \Delta_i - \Delta_{i-1} \tag{3}$$

The lemma shows that ALLOC is optimal—ignoring the remainder.

LEMMA 4.4. $\sum_{i=1}^{j} d_i \geq \sum_{i=1}^{j} c_i^*$ *for all* $1 \leq j < n$.

Proof. By contradiction. Suppose $\sum_{i=1}^{l} d_i < \sum_{i=1}^{l} c_i^*$ for $l < n$, and suppose that l is the smallest such index, i.e., using Eq. (2), $\Delta_l < 0$ and $\Delta_j \geq 0$ for all $0 \leq j < l$.

Hence, by Eq. (3), we have that $d_l < c_l^* - \Delta_{l-1}$, which implies, by integrality, that $d_l \leq c_l^* - \Delta_{l-1} - 1$. Therefore, since $T_l^* < (k+1)l$ by (iii) of Lemma 4.2, we have

$$\begin{aligned}
U_l &= U_{l-1} + ld_l \\
&\leq \sum_{i=1}^{l-1} id_i + lc_l^* - l\Delta_{l-1} - l \\
&= \sum_{i=1}^{l-1} i\left(c_i^* + \Delta_i - \Delta_{i-1}\right) + lc_l^* - l\Delta_{l-1} - l \\
&= \sum_{i=1}^{l-1} ic_i^* + \sum_{i=0}^{l-2}(i\Delta_i - (i+1)\Delta_i) + (l-1)\Delta_{l-1} + lc_l^* - l\Delta_{l-1} - l \\
&= T_l^* - \sum_{i=1}^{l-1} \Delta_i - l \\
&\leq T_l^* - l \\
&< (k+1)l - l \\
&= kl \ ,
\end{aligned}$$

i.e., $U_l < kl$, contradiction to Theorem 3.1, since $l < n$. □

We now prove that $\mathcal{S}$ is optimal even when considering the remainder.

LEMMA 4.5. $\sum_{i=1}^{m} c_i \leq \sum_{i=1}^{n} d_i$.

Proof. We consider two cases. If $n > m$ then using Lemma 4.4, and the fact that $d_n \geq 1$ by definition of n, we get

$$\sum_{i=1}^{m} c_i \ \leq \ \sum_{i=1}^{m} c_i^* + 1 \ \leq \ \sum_{i=1}^{n-1} d_i + 1 \ \leq \ \sum_{i=1}^{n} d_i \ ,$$

and we are done for this case.

So suppose $n \leq m$. Let $b = \sum_{i=n}^{m} c_i - d_n$. We prove the lemma by showing that $\sum_{i=1}^{n-1}(d_i - c_i) \geq b$. First, note that since by Lemma 4.4 we have $\Delta_i \geq 0$ for $i \leq n-1$, and using Eq. (3), we get

$$\begin{aligned}
U_{n-1} - T_{n-1}^* &= \sum_{i=1}^{n-1} i(d_i - c_i^*) \\
&= \sum_{i=1}^{n-1} i(\Delta_i - \Delta_{i-1}) \\
&= (n-1)\Delta_{n-1} - \sum_{i=0}^{n-2} \Delta_i \\
&\leq (n-1)\Delta_{n-1} \ .
\end{aligned}$$

On the other hand, since $U_n = N = T_m$, we get

$$\begin{aligned}
U_{n-1} &= U_n - nd_n \\
&= T_m - n\left(\sum_{i=n}^{m} c_i - b\right)
\end{aligned}$$

$$\begin{aligned} &\geq T_m - \sum_{i=n}^{m} ic_i + bn \\ &= T_{n-1} + bn \ . \end{aligned}$$

Therefore, it follows from Eq. (2) that

$$\begin{aligned} \sum_{i=1}^{n-1}(d_i - c_i^*) &= \Delta_{n-1} \\ &\geq \left\lceil \frac{U_{n-1} - T_{n-1}^*}{n-1} \right\rceil \\ &\geq \left\lceil \frac{T_{n-1} + bn - T_{n-1}^*}{n-1} \right\rceil \end{aligned} \tag{4}$$

We now consider two subcases. If $T_{n-1} = T_{n-1}^*$ then $c_i = c_i^*$ for $1 \leq i \leq n-1$, and Eq. (4) reduces to

$$\sum_{i=1}^{n-1}(d_i - c_i) \geq \left\lceil \frac{nb}{n-1} \right\rceil \geq b \ ,$$

and we are done for this subcase.

Otherwise, $T_{n-1} - T_{n-1}^* \geq 1$ and therefore

$$\begin{aligned} \sum_{i=1}^{n-1}(d_i - c_i) &\geq \sum_{i=1}^{n-1}(d_i - c_i^*) - 1 \\ &\geq \left\lceil \frac{T_{n-1} + bn - T_{n-1}^*}{n-1} \right\rceil - 1 \\ &\geq \left\lceil \frac{bn+1}{n-1} \right\rceil - 1 \\ &\geq b \ , \end{aligned}$$

and the proof of Lemma 4.5 is complete. □

4.2. The number of coins allocated by ALLOC**.** Theorem 4.1 proved that the number of coins in the solution produced by ALLOC is optimal. We now give a tight bound on that number in terms of N and k. There is a nice interpretation to the bound: it says that the worst penalty for having N and k which are not "nice" is equivalent to requiring an extra payment (i.e., solving the $(k+1)$-payment problem) for "nice" N and k. We remark that we do not know of any direct reduction which proves this result.

First, we prove an upper bound on m, which is slightly sharper than the general bound of Lemma 3.2.

LEMMA 4.6. *The largest denomination of a coin generated by* ALLOC *satisfies* $m \leq \left\lceil \frac{N}{k+1} \right\rceil$.

Proof. By (iii) of Lemma 4.2 we have $T_i^* < (k+1)i$, and in particular

$$m > \frac{T_m^*}{k+1} \tag{5}$$

By (ii) of Lemma 4.2 we have that $T^*_{m-1} \geq k(m-1)$, or that $m \leq \frac{T^*_{m-1}}{k} + 1$. Since $c_m \geq 1$, we have that $T^*_{m-1} \leq T^*_m - m$. Using also Eq. (5), we get that

$$\begin{aligned} m &\leq \frac{T^*_{m-1}}{k} + 1 \\ &\leq \frac{T^*_m - m}{k} + 1 \\ &< \frac{T^*_m - \frac{T^*_m}{k+1}}{k} + 1 \\ &= \frac{T^*_m}{k+1} + 1 \\ &\leq \frac{N}{k+1} + 1 \ , \end{aligned}$$

i.e., $m < \frac{N}{k+1} + 1$, and by integrality, $m \leq \left\lceil \frac{N}{k+1} \right\rceil$. □

THEOREM 4.7. *The number of coins in the solution produced by* ALLOC *is at most* $(k+1)H_{\lceil N/(k+1)\rceil} \approx (k+1)\ln\frac{N}{k+1}$.

Proof. By (iii) of Lemma 4.2, and by integrality, $T^*_i \leq (k+1)i-1$ for all $1 \leq i \leq m$. This, in conjunction with Lemmas 3.7 and 4.6, implies that

$$\begin{aligned} \sum_{i=1}^{m} c^*_i &= \sum_{i=1}^{m-1} \frac{T^*_i}{i(i+1)} + \frac{T^*_m}{m} \\ &\leq \sum_{i=1}^{m-1} \frac{i(k+1)-1}{i(i+1)} + k + 1 - \frac{1}{m} \\ &= \sum_{i=1}^{m-1} \frac{i(k+1)}{i(i+1)} - \sum_{i=1}^{m-1} \frac{1}{i(i+1)} + k + 1 - \frac{1}{m} \\ &= (k+1)(H_m - 1) + k \\ &\leq (k+1)H_{\lceil N/(k+1)\rceil} - 1 \ , \end{aligned}$$

and therefore, $\sum_i c_i \leq \sum_i c^*_i + 1 \leq (k+1)H_{\lceil N/(k+1)\rceil}$, as required. □

The upper bound given by theorem 4.7 is the best possible in general, as proven in the following theorem.

THEOREM 4.8. *For any natural numbers* n_1, n_2*, there are infinitely many* $N > n_1, k > n_2$ *such that any solution for the* k*-payment problem for budget* N *requires at least* $(k+1)H_{\lceil N/(k+1)\rceil}$ *coins.*

Proof. Choose a natural number m such that $m! - 1 > n_2$ and $m \cdot m > n_1$, and set $k = m! - 1$ and $N = 1 + \sum_{i=1}^{m} \lceil k/i \rceil \cdot i = m(k+1)$. For these N and k, we have that the algorithm produces largest denomination $m = \lceil N/(k+1) \rceil$, and $c_i = (k+1)/i$ for $1 \leq i \leq m$, and $c_i = 0$ otherwise. It follows that the number of coins in this case is precisely $(k+1)H_{\lceil N/(k+1)\rceil}$. □

5. Generalization to a restricted set of denominations. We now turn our attention to the restricted denomination case. In this setting, we are given a set $\mathcal{D}$ of natural numbers, and the requirement is that in any solution, $c_i = 0$ if $i \notin \mathcal{D}$.

To get an optimal algorithm for an arbitrary set of denominations which allows for a solution,[1] we follow the idea of algorithm ALLOC: ensure, with the least number

[1] Observe that the problem is solvable if and only if $1 \in \mathcal{D}$: if there are no 1-coins then certainly

```
GALLOC(N, k, 𝒟)
1  c_i ← 0 for all i
2  t ← 0
3  i ← 0
4  while N − t ≥ ī
5      do i ← ī
6         j ← ī − 1          ▷ j is the largest den. smaller than the next one in 𝒟
7         if t < kj                                  ▷ ensure T_l ≥ kl for all l up to j
8                then  if ⌊(N−t)/i⌋ ≥ ⌈(kj−t)/i⌉      ▷ check if budget is exhausted
9                      then  c_i ← ⌈(kj−t)/i⌉          ▷ if not, add coins
10                           t ← t + ic_i
11                     else  goto 12
12 if N > t
13   then  𝒟′ ← {d | d ∈ 𝒟, d ≤ i}
14         add MAKECHANGE(N − t, 𝒟′)
```

FIG. 4. *Algorithm for optimal solution of the k-payment problem with allowed denominations $\mathcal{D}$.*

of coins, that $T_i \geq ik$ for all $1 \leq i < m$. The treatment of the remainder is more complicated here, but has the same motivation: add the remainder with the least possible number of coins. We also have to make sure that the invariant maintained by the main loop is not broken, so we restrict the remainder allocation to use only denominations which were already considered. In fact, remainder allocation is precisely the *change-making* problem, solvable by dynamic programming. (For some allowed denomination sets [5, 14], the greedy strategy works too.) The main algorithm GALLOC is given in Figure 4. For completeness, we also include, in Figure 5, a description of a dynamic programming algorithm for optimal change-making. In GALLOC, and in the analysis, we use the following additional notation.

- $\underline{i} = \max_{j \in \mathcal{D}} \{j \mid j < i\}$, the largest allowed denomination smaller than i.
- $\overline{i} = \min_{j \in \mathcal{D}} \{j \mid j > i\}$, the smallest denomination larger than i.

We now prove that GALLOC is correct and that the number of coins it allocates is optimal. We remark that the general arguments are similar to those for the unrestricted case, but are somewhat more refined here.

THEOREM 5.1. *Let $\mathcal{S}$ denote the solution produced by* GALLOC. *Then $\mathcal{S}$ solves the k-payment problem with allowed denominations $\mathcal{D}$ using the least possible number of coins.*

Proof. Follows from Lemmas 5.3 and 5.8 below. □

We again use a loop invariant to capture the important properties of the algorithm.

LEMMA 5.2. *Whenever* GALLOC *executes line 4, the following assertions hold.*

(i) $t = \sum_{j=1}^{i} jc_j \leq N$.
(ii) if $N - t \geq \overline{i}$ *then* $t \geq jk$, *where* $j = \overline{i} - 1$.
(iii) if $i > 0$ *then* $t < jk + i$.

we cannot satisfy any payment request of value 1; and if 1 is allowed, then the trivial solution of N 1-coins works for all k.

```
MakeChange(R, 𝒟)
1  for i ← 1 to R
2      do if i ∈ 𝒟
3           then M[i] ← {i}
4           else M[i] ← ∅
5  while M[R] = ∅
6      do for i ← R downto 1
7       do if M[i] = ∅
8           then for all j ∈ 𝒟 s.t. j < i
9                do if M[i − j] ≠ ∅
10                    then M[i] ← M[i − j] ∪ {j}
11 return M[R]
```

FIG. 5. *Dynamic programming algorithm for finding least number of coins whose sum is R units using denomination set $\mathcal{D}$. M is an array of multisets.*

Proof. Line 4 can be reached after executing lines 1–3 or after executing the loop of lines 5–10. In the former case $t = i = 0$ and the lemma holds trivially.

In the latter case, let t_0 denote the value of the variable t before the last execution of the loop. If lines 9 and 10 are not executed, then (i) holds trivially. If they are executed, then $t = t_0 + ic_i$ and $t \le \left\lfloor \frac{N-t_0}{i} \right\rfloor \cdot i + t_0 \le N$, hence (i) holds after the loop is executed. Then, whether or not lines 9 and 10 are executed $t \le t_0 + \left\lceil \frac{kj-t_0}{i} \right\rceil \cdot i < t_0 + (kj - t_0 + i) = jk + i$, and thus (iii) is true after the execution of the loop.

Finally, we show (ii) holds after executing the loop. If line 8 is not executed, or if line 11 is executed then (ii) holds trivially. Otherwise lines 9–10 are executed, and $c_i = \left\lceil \frac{kj-t}{i} \right\rceil$, hence $t = t_0 + \left\lceil \frac{kj-t_0}{i} \right\rceil \cdot i \ge t_0 + kj - t_0 = kj$, as required. □

The correctness of GAlloc is proven in the next lemma. Following the conventions of §4, we denote by $\mathcal{S}^*$ the allocation produced by GAlloc before line 12. We denote by c_i^* and T_i^* the values in $\mathcal{S}^*$ corresponding to c_i and T_i in $\mathcal{S}$, respectively.

LEMMA 5.3. *$\mathcal{S}$ solves the k-payment problem with allowed denominations $\mathcal{D}$.*

Proof. By (i) of Lemma 5.2, in conjunction with lines 12–14 of the code, we have that upon completion of the algorithm, $\sum_i ic_i = N$. Let m be the highest denomination allocated by the algorithm. Then, by (ii) of Lemma 5.2, and the fact that MakeChange does not use coins with higher denominations (line 13), we have that for all $1 \le i < m$, $T_i \ge T_i^* \ge k(\overline{i} - 1) \ge ki$. Therefore, by Theorem 3.1, $\mathcal{S}$ solves the k-payment problem. □

We now turn to prove optimality of the algorithm. The analysis proceeds similarly to that of §4: We first show that the allocation of the coins up to the remainder (i.e., just before line 12 is executed), is optimal. We then prove that the handling of the remainder is optimal as well. The proof here is complicated by the fact that in order to ensure correctness, GAlloc allocates the remainder by using only denominations which were already considered in the main loop, and hence it is not immediately clear that the remainder is allocated optimally.

For the remainder of this section, fix an arbitrary "competitor" solution $\mathcal{T}$. We define for $\mathcal{T}$ notions analogous to those defined for the solution $\mathcal{S}$ produced by GAlloc.

- d_i is the number of i-coins in $\mathcal{T}$.
- U_i is the budget allocated in $\mathcal{T}$ using coins of denomination i or less: $U_i =$

$\sum_{j=1}^{i} id_i$.

- n is the largest denomination of a coin in $\mathcal{T}$: Formally, $n = \max\{i \mid d_i > 0\}$.

To facilitate treatment of the remainder, we fix an arbitrary subsolution $\mathcal{T}^*$ of $\mathcal{T}$ which solves the k-payment problem for budget T^*_m. (This is possible since a solution for budget N is also a solution for any budget smaller than N.) We use the following additional definitions.

- $U^*_i = \sum_{j=1}^{i} jd^*_j$. (Analogous to T^*_i in $\mathcal{S}$).
- n^* is the largest denomination in $\mathcal{T}^*$.
- m^* is the largest denomination allocated up to line 12 by GAlloc. Hence, $T^*_{m^*} = T^*_m$.

Finally, we define $\Delta_0 = 0$ and $\Delta_i = \Delta_{i-1} + d^*_i - c^*_i$ for $i > 0$. As before, by the definitions, for all $i \geq 0$ we have

$$\Delta_i = \sum_{j=1}^{i} d^*_j - \sum_{j=1}^{i} c^*_j \tag{6}$$

$$d^*_i = c^*_i + \Delta_i - \Delta_{i-1} \tag{7}$$

We start by proving that GAlloc produces an optimal solution—ignoring the remainder.

Lemma 5.4. *For all* $1 \leq j < n^*$, $\quad \sum_{i=1}^{j} d^*_i \geq \sum_{i=1}^{j} c^*_i$.

Proof. Suppose $\sum_{i=1}^{l} d^*_i < \sum_{i=1}^{l} c^*_i$ for $l < n^*$, and that l is the smallest such index, i.e., using Eq. (6), $\Delta_l < 0$ and $\Delta_j \geq 0$ for all $0 \leq j < l$. Hence, by Eq. (7), we have that $d^*_l < c^*_l - \Delta_{l-1}$, which implies, by integrality, that $d^*_l \leq c^*_l - \Delta_{l-1} - 1$. Also, $T^*_l < k(\overline{l} - 1) + l$ by (iii) of Lemma 5.2. Therefore, since $\overline{l} - 1 < n^*$, we get

$$\begin{aligned}
U^*_{\overline{l}-1} &= U^*_{l-1} + ld^*_l \\
&\leq \sum_{i=1}^{l-1} id^*_i + lc^*_l - l\Delta_{l-1} - l \\
&= \sum_{i=1}^{l-1} i\,(c^*_i + \Delta_i - \Delta_{i-1}) + lc^*_l - l\Delta_{l-1} - l \\
&= \sum_{i=1}^{l-1} ic^*_i + \sum_{i=0}^{l-2} (i\Delta_i - (i+1)\Delta_i) \\
&\qquad + (l-1)\Delta_{l-1} + lc^*_l - l\Delta_{l-1} - l \\
&= T^*_l - \sum_{i=1}^{l-1} \Delta_i - l \\
&\leq T^*_l - l \\
&< k(\overline{l} - 1) + l - l \\
&= k(\overline{l} - 1)\ ,
\end{aligned}$$

or $U^*_{\overline{l}-1} < k(\overline{l} - 1)$, a contradiction to Theorem 3.1, since $\overline{l} - 1 < n^*$. □

Corollary 5.5. $n^* \leq m^*$.

Proof. Suppose not. Then, by Lemma 5.4, $U^*_{n^*} = \sum_{i=1}^{n^*} id^*_i > \sum_{i=1}^{m^*} id^*_i \geq \sum_{i=1}^{m^*} ic^*_i = T^*_{m^*}$ which is a contradiction to the fact that $U^*_{n^*} = T^*_m = T^*_{m^*}$. □

Next, we prove that no solution can use a denomination larger than the largest one used by $\mathcal{S}$.

LEMMA 5.6. $n \leq m$.

Proof. We consider two cases. Suppose first that line 12 is reached after testing the condition on line 4. Then $m^* = m$, and hence the remainder allocated by the dynamic algorithm is $N - T^*_m < \overline{m}$, hence $N - T^*_m \leq m$. Note that $n \leq \max(n^*, N - T^*_m)$. By Corollary 5.5, $n^* \leq m^*$, and therefore, $n \leq \max(m^*, m) = m$ and we are done for this case.

Suppose next that line 12 is reached from line 11. In this case we have

$$\left\lfloor \frac{N - T^*_{m^*}}{m} \right\rfloor < \left\lceil \frac{k(\overline{m} - 1) - T^*_{m^*}}{m} \right\rceil ,$$

and since $T^*_{m^*} = U^*_{n^*}$, we get

$$\left\lfloor \frac{N - U^*_{n^*}}{m} \right\rfloor < \left\lceil \frac{k(\overline{m} - 1) - U^*_{n^*}}{m} \right\rceil ,$$

and hence

$$\begin{aligned} \frac{N - U^*_{n^*} - m}{m} &= \frac{N - U^*_{n^*}}{m} - 1 \\ &< \frac{k(\overline{m} - 1) - U^*_{n^*}}{m} , \end{aligned}$$

or

$$N - m \quad < \quad k(\overline{m} - 1) . \tag{8}$$

Now suppose for contradiction that $n > m$. Then at least one coin of denomination $n > m$ is allocated by $\mathcal{T}$, hence $N \geq n + U_m > m + U_m$, or $U_m < N - m$. Since $U_{\overline{m}-1} = U_m$, we get from Eq. (8) that $U_{\overline{m}-1} < k(\overline{m} - 1)$, a contradiction to Theorem 3.1 since $\overline{m} - 1 < \overline{m} \leq n$. □ It follows that the remainder is allocated optimally (for any choice of a subsolution $\mathcal{T}^*$).

COROLLARY 5.7. $\sum_{i=1}^{n} d_i - \sum_{i=1}^{n} d^*_i \quad \geq \quad \sum_{i=1}^{m} c_i - \sum_{i=1}^{m} c^*_i$.

Proof. By Lemma 5.6, the amount of $N - T^*_m$ must be allocated in $\mathcal{T}$ using only denominations used in $\mathcal{S}$. The statement therefore follows from the optimality of dynamic programming used in MAKECHANGE. □

We can now prove optimality of the full solution $\mathcal{S}$.

LEMMA 5.8. $\sum_{i=1}^{m} c_i \leq \sum_{i=1}^{n} d_i$.

Proof. By Corollary 5.7, it is sufficient to prove that the number of coins in $\mathcal{T}^*$ is at least as much as in $\mathcal{S}^*$. If $n^* \geq m^*$, then we are done by Lemma 5.4. It remains to consider the case of $n^* < m^*$. We do it similarly to Lemma 4.5: define $b = \sum_{i=n^*}^{m^*} c^*_i - d^*_{n^*}$. With this notation, it is sufficient to show that $\sum_{i=1}^{\underline{n}^*} (d^*_i - c^*_i) \geq b$.

By Lemma 5.4 we have $\Delta_i \geq 0$ for $i \leq \underline{n}^*$, and using Eq. (7) we get

$$U^*_{\underline{n}^*} - T^*_{\underline{n}^*} = \sum_{i=1}^{\underline{n}^*} i(d^*_i - c^*_i)$$

$$
\begin{aligned}
&= \sum_{i=1}^{\underline{n}^*} i(\Delta_i - \Delta_{i-1}) \\
&= \underline{n}^* \Delta_{\underline{n}^*} - \sum_{i=0}^{\underline{n}^*-1} \Delta_i \\
&\le \underline{n}^* \Delta_{\underline{n}^*} \ .
\end{aligned}
$$

Also, since $U^*_{n^*} = T^*_{m^*}$, we have

$$
\begin{aligned}
U^*_{\underline{n}^*} &= U^*_{n^*} - n^* d_{n^*} \\
&= T^*_{m^*} - n^* \left(\sum_{i=n^*}^{m^*} c^*_i - b \right) \\
&\ge T^*_{m^*} - \sum_{i=n^*}^{m^*} i c^*_i + b n^* \\
&= T^*_{\underline{n}^*} + b n^* \ .
\end{aligned}
$$

Hence, $n^* \Delta_{\underline{n}^*} \ \ge \ \underline{n}^* \Delta_{\underline{n}^*} \ \ge \ U^*_{\underline{n}^*} - T^*_{\underline{n}^*} \ \ge \ b n^*$, i.e., $\Delta_{\underline{n}^*} \ge b$. Thus, from Eq. (6) we get

$$
\sum_{i=1}^{\underline{n}^*} (d^*_i - c^*_i) \ \ = \ \ \Delta_{\underline{n}^*} \ \ \ge \ \ b \ ,
$$

as required. □

6. Conclusion. Motivated by electronic cash, we have specified and analyzed the *k-payment problem* in this paper. It is obvious that the problem is valid in conventional cash models as well: the main requirement is that the payments are made exactly, with no change given back (i.e., no cash given back from the vendor to the customer). We have given a simple "binary" solution where the denominations used are powers of 2, and an exact optimal solution for the general case, where the set of allowed denominations is arbitrary. If all denominations are allowed, the number of coins is about $k \ln(N/k)$.

We close this paper with a few open problems we find interesting.

- In the course of the analysis we have used a special kind of an approximation of the harmonic series, in which the sum of the first i elements, for each i, is the smallest possible value above $k \cdot i$. Are there other applications of the harmonic approximation used in this paper?
- Can the results be extended to the case where denominations may be negative?
- It is easy to see that the running time of Algorithms Alloc and GAlloc—without the invocation of MakeChange—is proportional to the largest denomination they allocate, and hence it is $O(N/k)$ by Lemma 3.2. For Algorithm MakeChange, it is not difficult to see that its running time is bounded by $O(N^2/k)$. Can the computational complexity of the algorithms in this paper be improved?
- Is there a direct reduction proving the results of §4.2?

Acknowledgments. We thank Mike Saks, Richard Stanley and Yishay Mansour for helpful discussions, Eric Bach for bringing the postage stamp problem to our attention, Rakesh Verma for providing us with a copy of [14], and Agnes Chan and the anonymous referees for valuable suggestions and comments.

REFERENCES

[1] S. Brands. Untraceable off-line cash in wallets with observers. In *Advances in Cryptology: Proc. of Crypto '93*, pages 302–318. Springer-Verlag (LNCS 773), 1993.

[2] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proc. of Crypto '82*, pages 199–203. Plenum Press N. Y., 1983.

[3] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Advances in Cryptology: Proc. of Crypto '88*, pages 319–327. Springer-Verlag, 1990.

[4] D. F. Hsu and X.-D. Jia. Construction of distributed loop networks. *SIAM J. Disc. Math*, 7(1):57–71, 1994.

[5] D. Kozen and S. Zaks. Optimal bounds for the change-making problem. *Theoretical Comput. Sci.*, 123:377–388, 1994.

[6] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. Wiley-Interscience Series in Discrete Mathematics. John Wiley and Sons, 1990.

[7] D. Naccache and D. M'Raïhi. Cryptographic smart cards. *IEEE Micro*, 16(3):14–23, June 1996.

[8] T. Okamoto. An efficient divisible electronic cash scheme. In *Advances in Cryptology: Proc. of Crypto '95*, pages 438–451. Springer-Verlag (LNCS 963), Aug. 1995.

[9] D. Pearson. A polynomail-time algorithm for the change-making problem. Technical Report TR94-1493, Cornell University, Ithaca, NY, June 1994.

[10] E. S. Selmer. On the postage stamp problem with 3 denominations. *Mathematica Scandinavica*, 56(2):105–116, 1985.

[11] E. S. Selmer. Associate bases in the postage stamp problem. *Journal of Number Theory*, 42(3):320–336, 1992.

[12] Y. Tsiounis. *Efficient Electronic Cash: New Notions and Techniques*. PhD thesis, College of Computer Science, Northeastern University, Boston, MA, 1997. See http://www.ccs.neu.edu/home/yiannis for information on availability.

[13] J. van Lint and R. Wilson. *A Course in Combinatorics*. Cambridge University Press, 1992.

[14] R. Verma and J. Xu. On optimal greedy change making. Technical Report UH-CS-94-03, Department of Computer Science, University of Houston, Apr. 1994.