ALGORITHMS FOR THE LOCAL AND THE GLOBAL POSTAGE STAMP PROBLEM

LÉO COLISSON PALAIS, JEAN-GUILLAUME DUMAS, ALEXIS GALAN, BRUNO GRENET, AND AUDE MAIGNAN

Abstract. We consider stamps with different values (denominations) and same dimensions, and an envelope with a fixed maximum number of stamp positions. The local postage stamp problem is to find the smallest value that cannot be realized by the sum of the stamps on the envelope. The global postage stamp problem is to find the set of denominations that maximize that smallest value for a fixed number of distinct denominations. The local problem is NP-hard and we propose here a novel algorithm that improves on both the time complexity bound and the amount of required memory. We also propose a polynomial approximation algorithm for the global problem together with its complexity analysis. Finally we show that our algorithms allow to improve secure multi-party computations on sets via a more efficient homomorphic evaluation of polynomials on ciphered values.

Contents

1. Introduction

The postage stamp problem combines number theory and combinatorial optimization. It arises from a simple and practical question: given a limited set of postage stamp denominations $A_k = \{a_1, a_2, \ldots, a_k\} \subseteq \mathbb{N}^k$ and at most s places to

2020 *Mathematics Subject Classification.* Primary 11B13, 11-04, 94A60.
Key words and phrases. Postage stamp problem, Secure multi-party computation.

stick the stamps, what is the first postage rate that cannot be formed? We formally define the maximal postage rate, namely the s-range, in Definition 1.1.

Definition 1.1. For a given positive integer s and a set of k positive and all different integers $A_k = \{a_1, a_2, \cdots, a_k\}$, with $a_1 < a_2 < \ldots < a_k$, the *s-range* is the largest integer n such that: $\forall i \leq n, \exists (\lambda_1, \ldots, \lambda_k) \in \mathbb{N}^k, \sum_{j=1}^{k} \lambda_j \leq s, i = \sum_{j=1}^{k} \lambda_j a_j$; n is then denoted $n_s(A_k)$. Then the *extremal s-range* is defined as $n_s(k) = \max_{A_k}(n_s(A_k))$ and a basis A_k satisfying $n_s(A_k) = n_s(k)$ is called an *extremal basis*.

We focus on the two main problems related to postage stamps, namely

Definition 1.2. The *local* postage stamp problem (LPSP) is the determination of the s-range for s stamps of a given basis of size k.

Definition 1.3. The *global* postage stamp problem (GPSP) is the determination of an extremal basis for given parameters k and s.

For example, if we choose $s = 2$ and $k = 3$, we have $n_2(3) = n_2(\{1, 3, 4\}) = 8$. Moreover $n_1(k) = k$ and $n_s(1) = s$. There is a simple upper bound on n:

Lemma 1.4. *For $A_k = \{a_1, \cdots, a_k\}$ a stamp basis, we have, for all s, $n_s(A_k) \leq sa_k$.*

Historically, Hans Rohrbach [22] was the first to mention the postage stamp problem in the case where $s = 2$. Then, a lot of effort was devoted to the search for extremal bases [22, 15, 21, 23, 26, 24, 16, 13, 14, 25, 4, 19, 32]. For instance, [5] provides a survey of known extremal s-ranges and extremal bases for small values of k and s. Then [27] shows that LPSP is NP-hard under Turing reductions, but can be solved in polynomial time if k is fixed. Nonetheless, it is possible to give polynomial-time approximation algorithms that compute interesting basis; even if evaluating exactly their s-range can be hard.

Contributions. The goal of this paper is to provide efficient *algorithms* for both the local and the global problems. We first propose a novel algorithm, asymptotically and practically better than existing methods for the local problem. In particular, with respect, e.g., to Mossige's s-range [17], we were able, first, to remove an asymptotic dependency in s, and, second, to also remove an s factor in the required memory. Then, we survey the different known methods that can compute *good* basis for the global problem: this means that for a given number of denominations k and positions s, we obtain a basis of denominations with an s-range with the same asymptotics as the (unknown) extremal basis. We show that a recursive divide & conquer algorithm, in which we combine smaller basis, is the best *polynomial* strategy, among these, to generate such good bases. We then prove that, asymptotically, the best recursion is where the smaller bases are balanced. For some basis, however, if one can afford more computational effort, there are examples where a dynamic programming, for instance, can find better bases, by choosing the best cut.

We have implemented all the different algorithms in the GStamps library[1].

Finally, we propose a way to improve in practice some cryptographic protocols relying on *private* polynomial evaluation, where the computations are made using fully homomorphic encryption.

[1] https://github.com/jgdumas/GStamps

Paper organization. In Section 2, we give two novel algorithms for the local problem that improve on the time complexity bound and on the amount of required memory. Then, in Section 3, we present a compared analysis of the Fibonacci sequence, the Alter and Barnett sequence, and the recursive divide & conquer algorithm for the approximation of the global stamp problem. Finally, in Section 4 we show how good approximation bases can speed up secure multi-party computations on sets via a more efficient homomorphic evaluation of polynomials on ciphered values.

2. Range algorithms for large envelopes

2.1. Mossige s-range algorithm.

For the sake of completeness we give in Algorithm 1, a version of the algorithm of [17] computing the s-range of a basis.

Algorithm 1 Mossige s-range [17]

Input: Basis $A_k = \{a_1 = 1 < a_2 < \ldots < a_k\}$, stamps s.
Output: The s-range $n_s(A_k)$.

```
Let Reached = [0, ..., 0] ∈ {0,1}^{a_k s};
for j = 1 to k do
    Reached[a_j] ← 1;
end for
for d = 1 to s − 1 do
    for i = a_k d down-to d do                ▷ a_k d is the largest reached for now
        if Reached[i] == 1 then
            for j = 1 to k do
                Reached[i + a_j] ← 1;         ▷ now reached with up to one more stamp
            end for
        end if
    end for
end for
for i = 2 to a_k s do
    if Reached[i] == 0 then
        return i − 1;                         ▷ smallest value not reached
    end if
end for
return a_k s;
```

Roughly, the idea is to compute the values reached with each number of possible stamps from 1 to s: with each additional stamp, all the previous values are examined (at most sa_k), and the previously reached ones, added to one of a_1 to a_k, are then attainable. We thus have Proposition 2.1.

Proposition 2.1 ([17]). *Algorithm 1 is correct and requires $O(ks^2a_k)$ assignments.*

More precisely, the number of assignments (modifications of the binary table storing whether each value is so far attainable or not) is lower than $a_k \frac{s(s-1)}{2} k$, with $n_s(A_k) \leq sa_k$.

2.2. **Incremental LPSP algorithm.** We describe in Algorithm 2 a novel algorithm computing the range given a fixed set of stamp values $a_1 < \cdots < a_k$ and a maximum number of allowed stamps s. This algorithm outperforms [17] both asymptotically and in practice when s is large enough, and is also arguably simpler. A first version of our algorithm already runs in time $O(ksa_k)$ instead of $O(ks^2a_k)$. In the following section we will then propose a second version improving on this. Furthermore, together with a better asymptotic time complexity $O(kn)$, it will also require a smaller memory footprint. The memory requirements of the first version and of the s-range algorithm [17] is indeed $O(n)$, while we will then show how to reduce it to $O(a_k)$ instead.

First, our novel algorithm keeps two tables T and U where all the cells of T are initialized to a large enough number, say $s+1$, except for $T[0] = 0$. The semantics of this array is that we know how to generate the integer i using $T[i]$ stamps when $T[i] \leq s$, while it is (not yet) known how to generate i otherwise. Table U is initialized to 1, the goal being that at the end, if $T[i] < s$, $U[i]$ should contain the index j of the largest value a_j involved in any optimal decomposition of i. We maintain a cursor, starting at position 0: the cursor at position i means that we know the optimal number of stamps for all elements $\leq i$. We then gradually expand our knowledge: if we can generate i using $T[i]$ stamps, we can also use one extra stamp (i.e. $T[i]+1$ stamps in total) to generate the number $i + a_j$ for any $j \in \{U[i], \ldots, k\}$ (note that we start the loop at $U[i]$, hence saving us from testing small stamps). If this combination is better than the previously known combination to generate $i+a_j$ (i.e. it requires less stamps), we update its value. In other words, if $T[i+a_j] > T[i]+1$ we just need to compute $T[i+a_j] \leftarrow T[i]+1$ and $U[i+a_j] \leftarrow j$. After this operation, $T[i+1]$ must contain the optimal number of stamps to generate $i+1$, where $T[i+1] > s$ means that it is impossible to generate $i+1$ in less than s steps. It is also possible to show (see Proposition 2.2) that $U[i]$ contains the highest possible stamp involved in any optimal decomposition of i. This way, if $T[i+1] > s$, we found the first element that cannot be generated with at most s steps, hence the s-range is $1, \ldots, i$. Otherwise we can just restart this procedure from the new optimal construction of $i+1$ until it finishes (see Algorithm 2 for the details).

Proposition 2.2. *Algorithm 2 is correct and requires* $O(k \cdot n_s(A_k) + s \cdot a_k) = O(ksa_k)$ *assignments.*

Proof. The correctness of Algorithm 2 follows the intuition given in the introduction of Section 2.2. More precisely, before showing the optimality of $T[i]$, we note that a simple induction shows that the algorithm maintains at any time the following invariant: for any i, if $T[i] \leq s$, then it is possible to construct i using $T[i]$ stamps where the decomposition involve at least once the stamp $a_{U[i]}$ (or no stamp in the special case of 0), and where all other involved stamps are smaller or equal to $a_{U[i]}$: the initialization of T is done so that all items but $T[0]$ are set to $s+1$ hence maintaining this invariant (0 is made of 0 stamps), and the only instruction that modifies T is $T[i+a_j] \leftarrow T[i]+1$ when $T[i] < T[i+a_j] \leq s+1$, hence $T[i] < s$, so we can apply the induction hypothesis: by induction, if $T[i] < s$, it is possible to decompose i into $T[i]$ stamps, hence by adding one more stamp, we can generate $T[i+a_j]$ with $T[i]+1$ stamps. Moreover, by induction, $T[i]$ can be decomposed using stamps smaller or equal to $U[i]$, and since $j > i$ and the basis is assumed to be sorted, the decomposition of $i+a_j$ uses the stamp $a_j = a_{U[i+a_j]}$ and this stamp is the higher stamp in the decomposition. Hence, the invariant is maintained.

Algorithm 2 Incremental LPSP

Input: Basis $A_k = \{a_1 = 1 < a_2 < \ldots < a_k\}$, stamps s.
Output: The s-range $n_s(A_k)$.

```
T ← [0, s+1, s+1, ..., s+1] ∈ ℕ^{(s+1)a_k+1}        ▷ s+1 plays the role of ∞
U ← [1, 1, 1, ..., 1] ∈ ℕ^{(s+1)a_k+1}
i ← 0
repeat
  for j in U[i]..k do
    if T[i + a_j] > T[i] + 1 then
      T[i + a_j] ← T[i] + 1          ▷ Changes when we find a better way to obtain i + a_j
      U[i + a_j] ← j
    end if
  end for
  i ← i + 1
until T[i] > s
return i − 1
```

Next, we show by induction the optimality of this value, by showing that when starting the t-th iterations (indexing from 0) of the main loop, $T[i]$ for $i \leq t$ contains $s+1$ if it is impossible to generate them with less than s stamps, and otherwise it contains the optimal number of stamps that can be used to generate i, while $U[i]$ contains the index of the highest stamp involved in any possible optimal decomposition (and 1 when $i = 0$). This is trivial when $t = 0$ since $T[0] = 0$ as 0 can be generated via 0 stamps. Then, assume $t > 0$. We do now a case analysis:

- We first consider a first case where t can be decomposed optimally (in terms of number of stamps) into $t = \sum_i \lambda_i a_i$ for some λ_i's with $\sum_i \lambda_i \leq s$, and such that this decomposition maximizes the highest stamp a_l involved with a non-zero coefficient $\lambda_l > 0$ if multiple such optimal decomposition exist. Then $t - a_l$ can be generated via $(\sum_i \lambda_i) - 1$ stamps ($t - a_l = (\lambda_l - 1)a_l + \sum_{i \neq l} \lambda_i a_i$) and this decomposition is optimal (if $t - a_l$ could be generated from less stamps, then this directly gives a better decomposition in stamps for t by simply adding the stamp a_l to this better decomposition, which is absurd since the decomposition of t was optimal). Hence by induction, we know that after the $t - a_l$-th loop, we have $T[t] = T[t - a_l] + 1$ (it cannot be smaller otherwise we could again find a better decomposition of t which is absurd since we assumed it was optimal) corresponding to the optimal stamp decomposition of t. This value cannot be changed in later iterations: it cannot be increased because of the test $T[i + a_j] > T[i] + 1$, and it cannot be decreased otherwise this would directly lead to a better decomposition of t. Also, $T[t]$ cannot have been set to its final value before the $t - a_l$-th iteration of the loop, because if it was set to its optimal value during iteration $t - x$ with $x > a_l$, it means that we could extract from it an optimal decomposition involving the stamp x. This is absurd since $x > a_l$ and we picked the optimal decomposition that was maximizing the highest possible stamp. Hence, it means that $U[i]$ is set during the $t - a_l$-th iteration of the loop to a_l, the highest possible stamp involved in the decomposition, concluding our induction for this case.

- The second case is when t cannot be decomposed into s stamps or less. But thanks to the invariant shown in the first part of the proof, we already know that $T[t] = s + 1$, concluding this case analysis and this induction proof.

For the complexity bound, the main loop repeats until i equals $n+1$, and i is incremented by 1 at each step, hence it repeats $O(n)$ times, while doing k assignments during the *for* loop, i.e. overall $O(kn)$ assignments. But trivially $n \leq sa_k$ (the highest number that can be generated places the maximum number of stamps with the maximum value). Moreover, the creation of T requires $O(sa_k)$ assignments, hence the overall number of assignments is $O(ksa_k)$. □

2.3. **Adding a sliding window.** We describe in Algorithm 3 an improvement of Algorithm 2 that is more space and time efficient: the size of its memory, $O(a_k)$, scaling only with the maximum stamp value a_k, and not with the total range (that could be up to $s \cdot a_k$). We also reduce the time dependency from the upper bound sa_k to the actually reached n.

Algorithm 3 Sliding window incremental LPSP

Input: Basis $A_k = \{a_1 = 1 < a_2 < \ldots < a_k\}$, stamps s.
Output: The s-range $n_s(A_k)$.

```
l ← ⌈log(a_k + 1)⌉          ▷ The size of the array is a power of 2 to have efficient modulo
w ← 2^l − 1                 ▷ We compute x mod l via the bitwise "and": x & w, for efficiency reasons
T ← [0, s+1, s+1, ..., s+1] ∈ ℕ^{2^l}        ▷ s+1 plays the role of ∞
U ← [1, 1, 1, ..., 1] ∈ ℕ^{2^l}
i ← 0
repeat
  for j in U[i]..k do
    if T[(i + a_j) & w] > T[i & w] + 1 then
      T[(i + a_j) & w] ← T[i & w] + 1
      U[(i + a_j) & w] ← j
    end if
  end for
  T[i & w] ← s + 1          ▷ Reset the sliding window
  i ← i + 1
until T[i & w] > s
return i − 1
```

The improvement of Algorithm 3 over Algorithm 2 comes from the main observation that in Algorithm 2 all operations on the array lies in a window of size $a_k + 1$. Hence, we can only keep this window and discard the rest of the array by considering all indices modulo $a_k + 1$. To improve the efficiency of the algorithm, we additionally consider a larger window whose size is a power of 2. This way, modulo operations can be implemented via a simple bitwise "*and*". We note that this algorithm is also slightly more efficient it term of time due to the fact that it does not need to create a large table of size sa_k (a rough estimation of n). This way, the complexity of this algorithm scales with n instead of sa_k, which may be interesting when $n \ll sa_k$.

Proposition 2.3. *Algorithm 3 is correct and requires* $O(k \cdot n_s(k) + a_k) = O(ksa_k)$ *assignments.*

Proof. The correction of this algorithm is a direct consequence of the correction of Proposition 2.2, combined with the fact that in Proposition 2.2 after i iterations the loop only modified elements in the table whose index is smaller than $i + a_k$, and will never access again elements whose index is smaller than i. Hence, we can write the new values on top of the items that will never be read again (and initialize them as we go), which is exactly the point of doing a modulo with a modulus 2^l larger than $a_k + 1$, implemented via the bitwise *and* operation with $w = 2^l - 1$ for efficiency reasons.

The complexity bound is analog to Proposition 2.2, except that the creation of the original table does $O(a_k)$ assignments instead of $O(sa_k)$, while the loop still stops after n iterations doing k assignments at each iteration. □

2.4. **Comparison to the state of the art.** Table 1 compares the asymptotical complexity of our two algorithms (Algorithms 2 and 3) with [17].

TABLE 1. Comparison of asymptotic complexity

	[17]	Algorithm 2	Algorithm 3
Time	$O(ks^2a_k)$	$O(ksa_k)$	$O(ksa_k)$
Memory	$O(sa_k)$	$O(sa_k)$	$O(a_k)$

We also compare those three algorithms based on numerical simulations to also assert the practicality of our approach. [17] remains faster only for very small values of s (smaller than 5 in our simulations) as shown in Figure 1 for $s = 5$ and $s = 10$, or Figures 2 and 3 for various s. Note that the early termination condition of Selmer's Lemma (see [23, 26, 24]) can be added to all the presented range algorithms : when the condition of the Lemma is met during the computation, the range is then known [18, (10)]. For instance, Figure 2 suggests a speed-up growing linearly with the number of stamps, that reaches a threshold when Selmer's Lemma is used to early terminate the range computations.

The range algorithms are implemented in the library GStamps, and can be respectively used via `./bin/srange` for Algorithm 1, `./bin/reach` for Algorithm 2 and `./bin/krange` for Algorithm 3.

3. Polynomial-time approximations for the global postage stamp

This section is devoted to approximation algorithms for the global problem : Given k and s, compute a lower bound on $n_s(k)$ that is as close as possible to the correct (unknown) value. This goes through producing some basis $A_{k,s}$ with k denominations and proving a lower bound on $n_s(A_{k,s})$.

We review some existing constructions. Our starting point is the paper by Alter and Barnett [1]. They present, for all $k \geq s$, a *good* basis $A_{k,s}$. In particular, in the case $k = s$, $A_{k,s}$ is made of the first k even-indexed Fibonacci numbers. They provide the exact value for $n_s(A_{k,s})$. We extend their construction to the case $k \leq s$, and slightly improve it in the case $k \geq s$. Although they give the correct value for $n_s(A_{k,s})$, their proof only proves the lower bound. We bring a full proof

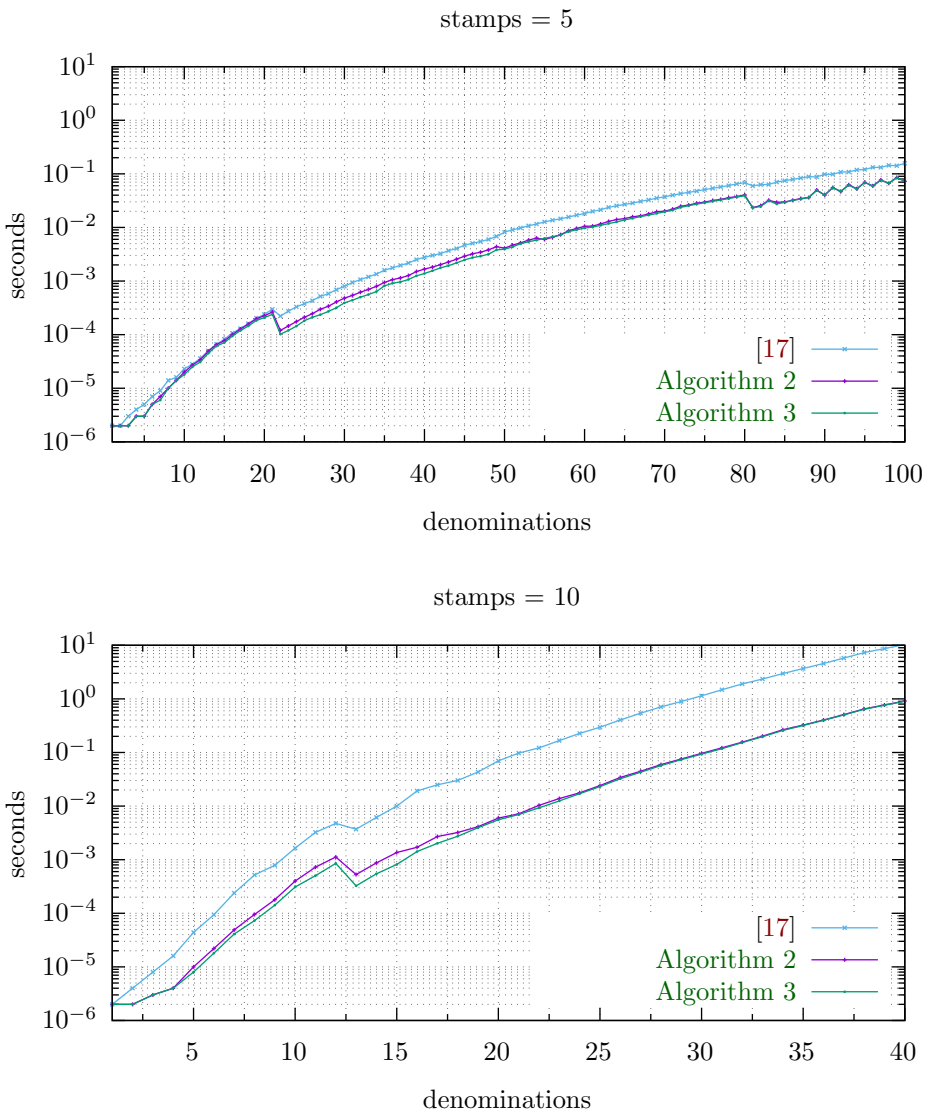

FIGURE 1. LPSP algorithms, $s = 5$ and $s = 10$

of the upper bound that applies both to their original algorithm and our improved version. We also analyse the asymptotics of $n_s(A_{k,s})$ in both cases.

We then develop and analyse an idea of Mrose [20] that builds a good basis $A_{k,s}$ with a recursive algorithm. We also analyse the effect of using a database of small good base cases. Combining several constructions and the database of base cases, we obtain the best known asymptotic algorithms as well as experimental validation of these results. Using our GStamps library, one can retrieve all the results presented in this part. Namely, our algorithms `fibo, alba, bala, geom` compute the s-ranges for (respectively) the Fibonacci basis (Definition 3.3), the Alter and Barnett

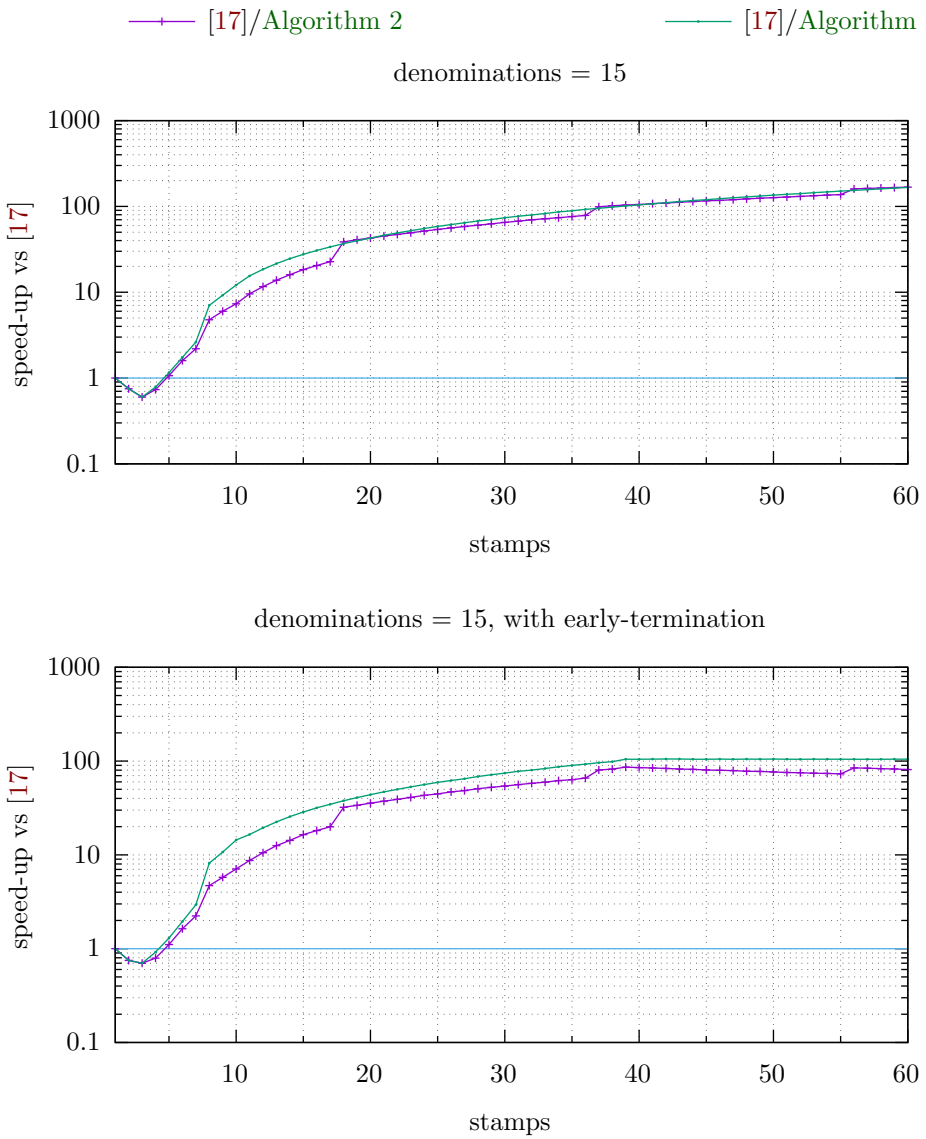

FIGURE 2. LPSP algorithms, $k = 15$, early terminated by Selmer's lemma [23] (effective for $s \geq 40$).

basis (Corollary 3.5), its more balanced variant (Definition 3.6) and the geometric basis (Remark 3.10), for chosen k, s. Our algorithm `basis` computes the best found basis for given k, s using the recursive algorithm and our database of base cases.

3.1. **Generalized Alter-Barnett's construction.** Alter and Barnett [1] propose a construction of a *good* basis for all $k \geq s$. Although they state the correct s-range, their proof is incomplete. Only the (easier) upper bound is proved. We provide

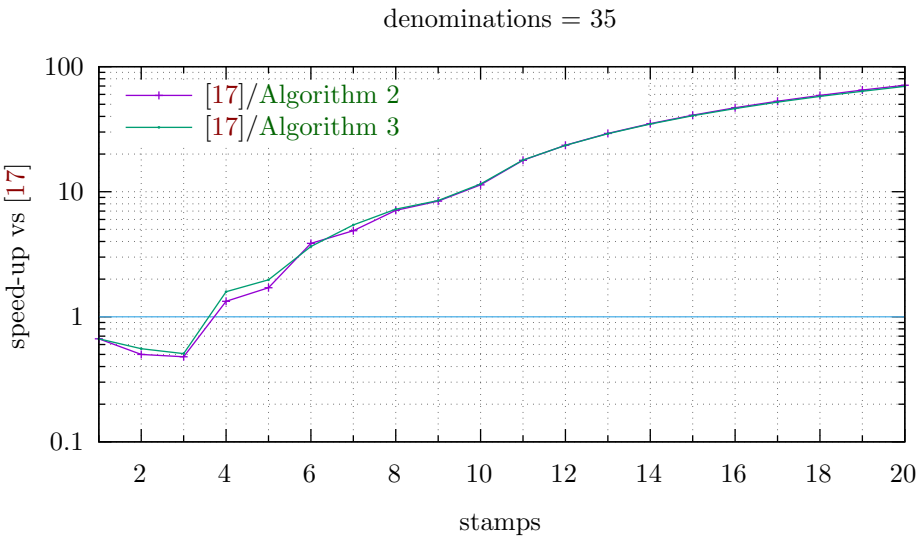

FIGURE 3. LPSP algorithms, $k = 35$

a complete proof of their result, and actually generalize it. As a consequence, we obtain a slightly better basis.

The construction of the stamp basis for s places is a union of s *blocks*, B_i. Each block is an arithmetic progression. For each $t \leq s$, the t-th blocks starts at the smallest value not reachable from the previous blocks using t stamps (that is $1+n_t(B_1 \cup \cdots \cup B_{t-1})$), and its common difference is the smallest value not reachable from the previous blocks using $(t-1)$ stamps (that is $1 + n_{t-1}(B_1 \cup \cdots \cup B_{t-1})$.

The following Definition 3.1 provides a direct construction not referring to $n_{t-1}(\cdot)$ nor $n_t(\cdot)$. It is illustrated on Figure 4, and the s-range of the construction is proved in Theorem 3.2.

Definition 3.1. Let $q_1, \ldots, q_s \geq 1$. Let $u_1, \ldots, u_s$ and $v_1, \ldots, v_s$ be integers defined by $u_1 = v_1 = 1$ and for $i \geq 1$, $v_{i+1} = u_i + q_i v_i$ and $u_{i+1} = u_i + (q_i - 1)v_i + v_{i+1}$.

Let $B_1, \ldots, B_s$ defined by $B_i = \{u_i + tv_i : 0 \leq t < q_i\}$ for $i \in [s]$. The *stamp basis associated to* $q_1, \ldots, q_s$ is

$$G(q_1, \ldots, q_s) = B_1 \cup \cdots \cup B_s.$$

Theorem 3.2. *Let* $q_1, \ldots, q_s \geq 1$ *and* $G = G(q_1, \ldots, q_s)$. *Then* $n_s(G) = u_s + q_s v_s - 1$.

Proof. Let $v_{s+1} = u_s + q_s v_s$, $u_{s+1} = u_s + (q_s - 1)v_s + v_{s+1}$ and for $1 \leq i \leq s$, $G_i = G(q_1, \ldots, q_i)$. We start by proving, by induction on i, that $n_i(G_i) \geq v_{i+1} - 1$ and $n_{i+1}(G_i) \geq u_{i+1} - 1$. This implies in particular the lower bound $n_s(G) \geq u_s + q_s v_s - 1$. The base case $i = 1$ is clear since $G_1 = \{1, \ldots, q_1\}$, $v_2 = 1 + q_1$ and $u_2 = 2q_1 + 1$. Assume now that $n_{i-1}(G_{i-1}) \geq v_i - 1$ and $n_i(G_{i-1}) \geq u_i - 1$ for some $i \geq 2$. Any integer $m < u_i$ can be written with i stamps from $G_{i-1} \subset G_i$. Consider an integer $m \in \{u_i, \ldots, v_{i+1} - 1\}$. Since $v_{i+1} = u_i + q_i v_i$, if we take the largest possible stamp $u_i + tv_i \leq m$ of G_i, $m - (u_i + tv_i) < v_i$. Therefore, $m - (u_i + tv_i)$

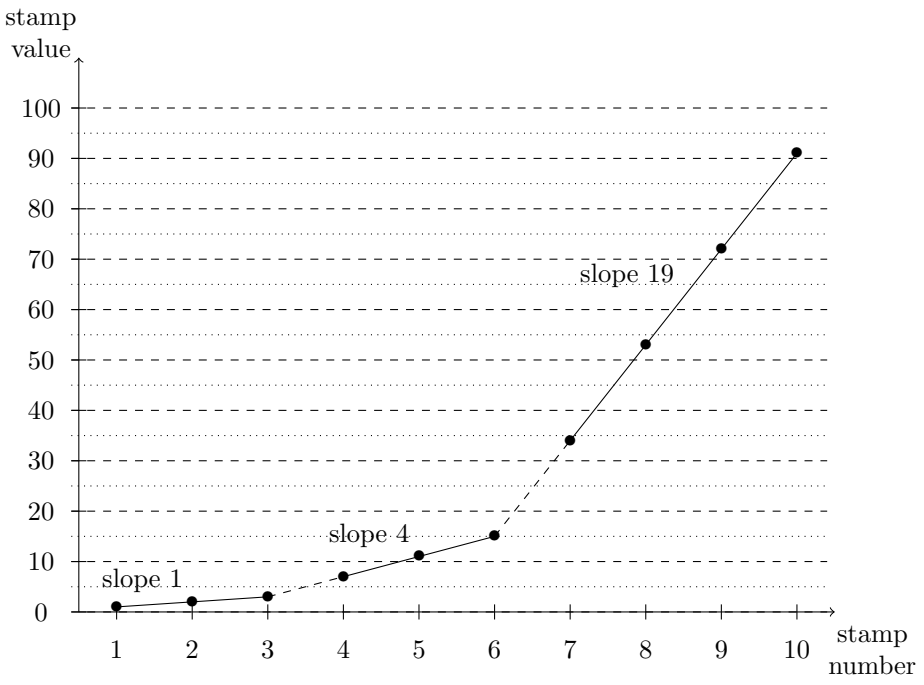

FIGURE 4. Representation of $G(3,3,4)$ with three blocks of stamps of denominations $1,2,3;7,11,15;34,53,72,91$, reaching $n_3 = 109$.

can be written using $(i-1)$ stamps from G_{i-1} and m can be written with i stamps from G_i. Thus $n_i(G_i) \geq v_{i+1} - 1$. Now take any $m \in \{v_{i+1}, \ldots, u_{i+1} - 1\}$. Since $u_{i+1} = u_i + (q_i - 1)v_i + v_{i+1}$, the same argument as before proves that using the largest possible stamp $u_i + tv_i \leq m$, $m - (u_i + tv_i) < v_{i+1}$. Therefore, $m - (u_i + tv_i)$ can be written with i stamps from G_i, and m with $(i+1)$ stamps from G_i. That is, $n_{i+1}(G_i) \geq u_{i+1} - 1$.

Let us now prove that $n_s(G_s) < v_{s+1} = u_s + q_s v_s$, that is v_{s+1} cannot be written as a sum of at most s stamps from G. A stamp in block i has value $u_i + tv_i$ for some t. Grouping all stamps in a same block, we can write the sum of their values as $a_i u_i + b_i v_i$. A sum corresponding to at most s stamps from G is $\sum_{i=1}^{s} a_i u_i + b_i v_i$ with $\sum_i a_i \leq s$ and $b_i \leq (q_i - 1)a_i$ for all i. We identify the sum with the sequence $((a_i, b_i))_i$.

We will prove that given such a sequence, we can produce a new sequence $((a'_i, b'_i))_i$, with the same sum, with the extra property that for every $i > 0$, $a'_i \leq 1$ and $a'_i + b'_i \leq q_i$. To this end, we use identities:

(1) $u_i + q_i v_i = v_{i+1}$;
(2) $2v_i - u_i = v_{i-1}$;
(3) $u_i - v_i = u_{i-1} + (q_{i-1} - 1)v_{i-1}$.

The first and third ones are the definition of the sequences. The second one combines the two other ones: $u_i = u_{i-1} + (q_{i-1} - 1)v_{i-1} + v_i$, and $u_{i-1} + q_{i-1}v_{i-1} = v_i$, whence $u_i = (v_i - v_{i-1}) + v_i$.

These relations allows us to define three rewriting rules on the sequence $((a_i, b_i))_i$ that do not change the value $\sum_i a_i u_i + b_i v_i$:

(1) if $a_i > 0$ and $b_i \geq q_i$, replace the pairs $(a_i, b_i), (a_{i+1}, b_{i+1})$ by $(a_i - 1, b_i - q_i), (a_{i+1}, b_{i+1} + 1)$ (using the first identity);
(2) if $a_i = 0$ and $b_i > q_i$, replace $(a_{i-1}, b_{i-1}), (a_i, b_i)$ by $(a_{i-1}, b_{i-1} + 1), (a_i + 1, b_i - 2)$ (using the second identity);
(3) if $a_i > 1$ and $b_i < q_i$, replace $(a_{i-1}, b_{i-1}), (a_i, b_i)$ by $(a_{i-1} + 1, b_{i-1} + (q_{i-1} - 1)), (a_i - 1, b_i + 1)$ (using the third identity).

We will apply these rules with $i > 0$. Note that if $i = s$, we may have to consider (a_{s+1}, b_{s+1}). This is possible by extending the sequences $(u_i)_i$ and $(v_i)_i$ with $v_{s+1} = u_s + q_s v_s$ and $u_{s+1} = u_s + (q_s - 1)v_s + v_{s+1}$. We can choose any very large value for q_{s+1} to never apply the first or second rule with $i = s + 1$.

Starting from a sum $\sum_i a_i u_i + b_i v_i$, our goal is to use the rewriting rules to obtain a sum where for all $i > 0$, $a_i \leq 1$ and $a_i + b_i \leq q_i$. The starting point is a sum where $\sum_i a_i \leq s$, and $b_i \leq (q_i - 1)a_i$ for each i. We call a sequence $((a_i, b_i))_{i \geq 0}$ *valid* if $a_i + b_i \leq \max(1, a_i)q_i$ and $\sum_i a_i \leq s$. In particular, the starting sequence is valid.

We choose the largest index j that violates the conditions, that is $a_j > 1$ or $a_j + b_j > q_j$. If $j > 1$, we apply the first of the three rules that applies to (a_j, b_j), and recurse. We stop when $a_i \leq 1$ and $a_i + b_i \leq q_i$ for all $i > 0$. We need to prove that this process terminates.

First assume that the first rule is applied to a valid sequence. Since this replaces (a_{j+1}, b_{j+1}) by $(a_{j+1}, b_{j+1} + 1)$, the sequence becomes invalid if $a_{j+1} + b_{j+1} = q_{j+1}$. (Recall that $a_{j+1} \leq 1$ by maximality of j.) If $a_{j+1} = 1$, the first rule is again applied to replace $(1, q_{j+1})$ by $(0, 0)$ and (a_{j+2}, b_{j+2}) by $(a_{j+2}, b_{j+2} + 1)$. The situation may repeat a few times, but after a few iterations the rewriting rule will encounter a pair (a_{j+t}, b_{j+t}) where either $a_{j+t} + b_{j+t} < q_{j+t}$ or $a_{j+t} = 0$. In the first case, the sequence is valid again. In the second case (with $b_{j+1} = q_{j+t} + 1$), the second rule is applied and the sequence becomes valid again too. In both cases, after these few steps, the sequence is valid and either j has decreased or a_j has decreased. Note also that to apply the first rule to a valid sequence, a_j has to be at least 2 (since $a_j = 1$ implies $b_j \leq q_j - 1$). After these steps, a_j remains non-zero. The same phenomenon occurs when the third rule is applied to a valid sequence. It replaces $(a_{j-1}, b_{j-1}), (a_j, b_j)$ by $(a_{j-1} + 1, b_{j-1} + (q_{j-1} - 1)), (a_j - 1, b_j + 1)$, with the condition that $a_j > 1$ and $b_j < q_j$. The sequence becomes invalid if $a_j = 2$ and $b_j = q_j - 1$. But then the pair becomes $(1, q_j)$ and the next rule to be applied is the first rule. As before, after a few steps, the sequence becomes valid again and either j or a_j has decreased, and a_{j-1} is non-zero. The second rule only applies to invalid sequences. By grouping the rewriting steps into *meta-steps*, we get a rewriting process where the sequence stays valid and the pair (j, a_j) decreases (for the lexicographic order) at each meta-step. The process thus stops with a valid sequence.

Assume therefore that $v_{s+1} = u_s + q_s v_s$ admits a representation with s stamps. Then $v_{s+1} = \sum_i a_i u_i + b_i v_i$ where $a_i \in \{0, 1\}$ and $0 \leq b_i < q_i$ for $i > 0$, and $\sum_i a_i \leq s$. After each meta-step, $a_i \neq 0$ for at least one index $i \leq j$. In particular, $\sum_{i=1}^{s} a_i u_i + b_i v_i > 0$ at the end of the process. Therefore, the pairs (a_i, b_i), $i > s$, must be zero. Assume otherwise that $(a_j, b_j) \neq (0, 0)$ for some $j > s$. The total sum is then at least $(a_j u_j + v_j b_j) + \sum_{i=1}^{s} a_i u_i + b_i v_i > a_j u_j + b_j v_j$. Since u_j, $v_j \geq v_{s+1}$,

the sum is $> v_{s+1}$, a contradiction. Thus $v_{s+1} = \sum_{i=1}^{s} a_i u_i + b_i v_i$, with $\sum_i a_i \leq s$. By validity of the sequence $a_i u_i + b_i v_i \leq u_i + (q_i - 1)v_i$ for all i. The largest possible sum is then $\sum_{i=1}^{s} u_i + (q_i - 1)v_i = \sum_{i=1}^{s} v_{i+1} - v_i = v_{s+1} - 1$. Therefore, v_{s+1} cannot be reached and $n_s(G_k) \leq v_{s+1} - 1$. □

As a special case, we get for $k = s$ the basis $G(1, \ldots, 1) = \{f_{2i}\}_{i \in [1,k]}$, where f_j is the jth Fibonacci number, with k-range $n_k(F_k) = f_{2k+1} - 1$. This basis can be used also if $k \leq s$ and we formally define it in Definition 3.3.

Definition 3.3. For k denominations and s places, with $k \leq s$, $F_k = \{f_{2i}\}_{i \in [1,k]}$ where f_j is the jth Fibonacci number, is a *Fibonacci basis*.

Corollary 3.4. *For $k \leq s$, $\left\lceil \frac{s}{\sqrt{5}} \right\rceil (1+\varphi)^k > n_s(F_k) > \left\lfloor \frac{s-k+1}{\sqrt{5}} \right\rfloor (1+\varphi)^k$ where φ denotes the golden ratio.*

Proof. For the upper bound, by Lemma 1.4 with basis F_k, we get $n_s(F_k) < s f_{2k}$. Then $f_{2k} = \left\lceil \frac{1}{\sqrt{5}} \varphi^{2k} \right\rceil = \left\lceil \frac{1}{\sqrt{5}} (1+\varphi)^k \right\rceil$. For the lower bound, use k stamps to reach any value below $f_{2k+1} - 1$. Then any value between f_{2k+1} and $f_{2k+1} - 1(s-k)f_{2k}$ is reached with $(s-k)$ times the largest stamp $a_k = f_{2k}$, and k more stamps. Thus $n_s(F_k) \geq f_{2k+1} + (s-k)f_{2k} - 1 > (s-k+1)f_{2k}$. The lower bound follows using again $f_j = [\frac{1}{\sqrt{5}} \varphi^j]$. □

We can now use Theorem 3.2 to build good bases for the global postage stamp problem when $k \geq s > 0$. Write k as a sum of s positive integers $q_1, \ldots, q_s$ and define the basis $G(q_1, \ldots, q_s)$. The original construction of Alter and Barnett [1] uses the Euclidean division $k = qs + r$ with $r < s$ to define the basis $G(q, \ldots, q, q+r)$. This basis satisfy $n_s(G(q, \ldots, q, q+r)) = \sum_{i=1}^{s} \binom{s+i}{2i} q^i + r \sum_{i=0}^{s-1} \binom{s+i-1}{2i} q^i$. Our generalization allows us to define better bases. In particular, more balanced bases have a larger s-range. For instance, with $k = 8$ and $s = 3$, their basis is $G(2, 2, 4)$ and its s-range is 62, while the basis $G(3, 3, 2)$ has s-range 71. More generally, the basis $G(q+1, \ldots, q+1, q, \ldots, q)$ with r copies of $(q+1)$ and $(s-r)$ copies of q has a larger s-range than $G(q, \ldots, q, q+r)$. Nevertheless the former is not always the best basis: For instance with $k = 7$ and $s = 5$, $n_5(G(2, 2, 1, 1, 1)) = 206$ while $n_5(G(2, 1, 2, 1, 1, 1)) = 207$. We shall now prove that, asymptotically, it is better to use bases with r copies of $(q+1)$ and $(s-r)$ copies of q.

Corollary 3.5. *Let $k \geq s > 0$ and write $k = qs + r$ with $r < s$ and let $\epsilon_q = \frac{1}{2}\sqrt{1 + 4/q} - \frac{1}{2}$, then:*

- $n_s(G(q, \ldots, q, q+r)) = \Theta((1 + q(1+\epsilon_q))^{s-1}(1 + (q+r)(1+\epsilon'))))$, *with $\epsilon' < \frac{2}{q}$;*
- $n_s(G(q+1, \ldots, q+1, q, \ldots, q) = \Theta((1 + (q+1)(1+\epsilon_{q+1}))^r (1 + q(1+\epsilon_q))^{s-r})$.

From Corollary 3.5, we see that the ratio in the dominant term is in both cases asymptotically close to $1 + q = 1 + \frac{k}{s}$. With $\epsilon_q = \frac{\sqrt{1+4/q}-1}{2}$, we thus obtain bases whose s-ranges are asymptotically given by the generic form of Equation (3.1).

$$\alpha_q \left(1 + \frac{k}{s}(1 + \epsilon_q)\right)^s. \tag{3.1}$$

Proof. Consider a basis $G(q, \ldots, q)$ and let $n_s = n_s(G(q, \ldots, q))$. Since $n_s = u_s + q v_s - 1$ and u_s and v_s are defined by recurrence relations, we obtain the following

induction relation

$$n_{s+2} = (q+2)n_{s+1} - n_s + q, n_0 = 0, n_1 = q. \quad \text{[1, Eq. (20)]} \tag{3.2}$$

By solving the general term of this recurrence relation exactly for indeterminates n_0 and n_1, one gets that the dominant term of $n_s(G(q,\dots,q,q+r))$ is:

$$\alpha_q(1+q(1+\epsilon_q))^s, \text{ with } \alpha_q = \frac{1}{1+2\epsilon_q}\left((n_0+1)\epsilon_q + \frac{n_1-n_0}{q}\right). \tag{3.3}$$

For a basis of form $G(q,\dots,q,q+r)$, we use Equation (3.3) until $s-1$, so that $n_{s-2} = \alpha_q\,(1+q(1+\epsilon_q))^{s-2}$ and $n_{s-1} = \alpha_q\,(1+q(1+\epsilon_q))^{s-1}$. Then use Equation (3.2) with a ratio $q+r$ to obtain $n_s = (q+r+2)n_{s-1} + (q+r) - n_{s-2} = n_{s-1}\left(1+(q+r)(1+\frac{1}{q+r}+\frac{1}{n_{s-1}}-\frac{n_{s-2}}{n_{s-1}})\right)$. This is asymptotically the first term of the Corollary, using $\epsilon' = \frac{1}{q+r}+\frac{1}{n_{s-1}} - \frac{1}{1+q(1+\epsilon_q)} < \frac{1}{q}+\frac{1}{n_{s-1}} < \frac{2}{q}$.

Similarly for the second bases, Equation (3.3) with $q \leftarrow q+1$ and $s \leftarrow r$, for the first r blocks, gives that $n_r = \Theta(n_{r-1}(1+(q+1)(1+\epsilon_{q+1})))$. Then use Equation (3.3) again for the next $(s-r)$ blocks, now with q and $(n_0,n_1) = (n_{r-1},n_r)$ from the previous blocks. This gives that $n_s = \Theta(n_{s-1}(1+q(1+\epsilon_q)))$ and, combining both approximations, that $n_s = \Theta((1+q(1+\epsilon_q))^{s-r}n_r) = \Theta((1+q(1+\epsilon_q))^{s-r}(1+(q+1)(1+\epsilon_q))^r)$. □

In the following, we thus balance the quotients in Definition 3.1 as suggested by the second equation of Corollary 3.5 and denote the associated basis as in Definition 3.6.

Definition 3.6. A *Balanced stamp basis* for k denominations and s places, with $k \geq s$, is given recursively via the Definition 3.1, for $k = qs+r$ obtained by Euclidean division, as: $G(\underbrace{q+1,\dots,q+1}_{r \text{ times}}, \underbrace{q,\dots,q}_{s-r \text{ times}})$.

Both basis of Corollary 3.5 are implemented in the library GStamps, respectively in the `./bin/alba` and `./bin/bala` programs.

3.2. **Recursive algorithm.** From two sub-bases A_{k_1}, B_{k_2} with respective s-ranges $n_{s_1}(A_{k_1})$ and $n_{s_2}(B_{k_2})$, using [20, Hilfssatz 2], we can build a basis C_k of size $k = k_1+k_2$ such that, for $s = s_1+s_2$: $n_s(C_k) \geq (n_{s_1}(A_{k_1})+1)(n_{s_2}(B_{k_2})+1)-1$. As this relation is true for any basis, we more precisely have the following Proposition 3.7.

Proposition 3.7. $n_{s_1+s_2}(k_1+k_2) \geq (n_{s_1}(k_1)+1)(n_{s_2}(k_2)+1)-1$

Proof. We recall the proof of [20]. Let $A_{k_1} = \{a_1,\cdots,a_{k_1}\}$, $B_{k_2} = \{b_1,\cdots,b_{k_2}\}$, $n_1 = n_{s_1}(A_{k_1})$ and $n_2 = n_{s_2}(B_{k_2})$. We define C_k as the concatenation of A_{k_1} and all the denominations of B_{k_2} multiplied by n_1+1, i.e. $C_k = A_{k_1} \| (n_1+1)B_{k_2}$. Then, $i(n_1+1)$ for $i \in [n_2]$ are generated with at most s_2 stamps, using the stamps from $(n_1+1)B_{k_2}$. We also can generate all $j \in [n_1]$ with at most s_1 stamps from A_{k_1}. Combined together, we can obtain all $i(n_1+1)+j$ for $i \in [n_2]$ and $j \in [n_1]$ with at most s stamps. Then, $n_s(C_k) \geq n_2(n_1+1)+n_1$. By taking extremal bases for A_{k_1} and B_{k_2}, we obtain that $n_s(C_k) \geq (n_{s_1}(k_1)+1)(n_{s_2}(k_2)+1)-1$. The result follows, as by definition $n_s(k) \geq n_s(C_k)$. □

The constructive proof of Proposition 3.7 thus provides a recursive divide & conquer algorithm computing a basis by recursively splitting both the number of

denominations and the number of stamps.[2] In the rest of this section, we analyze the behavior of this recursive divide & conquer polynomial-time algorithm. We start by determining the asymptotics, with the help of a simpler function in Lemma 3.8. This provides a lower bound when $k_1 = k_2$ and $s_1 = s_2$ are both powers of two, dividing both parameters by 2, until one of k or s is 1.

Lemma 3.8. *Let $f : \mathbb{N} \to \mathbb{N}$ such that $f(0) = T$ and, for $i \geq 0$, $f(i+1) = (f(i)+1)^2 - 1$. Then $f(i) = (1+T)^{2^i} - 1$.*

Proof. The equality holds for $i = 0$ since $f(0) = T = (1+T)^{2^0} - 1$. Then by induction hypothesis, $f(i+1) = ((1+T)^{2^i})^2 - 1 = (1+T)^{2^{i+1}} - 1$. □

Now, suppose that $2^{i+j} = k \geq s = 2^i$. We use a *midpoint construction*, cutting both parameters in halves. The recursive cuts stop when the number of stamps is 1, with $n_1(\frac{k}{s}) = \frac{k}{s}$. Combining Proposition 3.7 and Lemma 3.8 with $T = \frac{k}{s}$, we obtain the lower bound

$$n_s(k) \geq \left(1 + \frac{k}{s}\right)^s - 1. \tag{3.4}$$

This is similar to the asymptotics of Corollary 3.5, but slightly less interesting since there the common ratio is $(1 + q(1+\epsilon_q))$, indeed slightly better than $1 + q = \left(1 + \frac{k}{s}\right)$.

For the opposite case, namely $2^{i+j} = s \geq k = 2^i$, the recursion stops with 1 denomination, with $n_{\frac{s}{k}}(1) = \frac{s}{k}$. This results in the dual lower bound

$$n_s(k) \geq \left(1 + \frac{s}{k}\right)^k - 1. \tag{3.5}$$

This is now to be compared with Corollary 3.4 with the same exponent but a common ratio $(1+\varphi)$. We see that for small values of k the recursive algorithm will be better, and then Fibonacci's algorithm will be better when k is closer to s with ratio $\frac{s}{k} < \varphi$.

These bounds can actually be refined by stopping the recursion earlier, relying on the exact values known for small values of k and s.

Proposition 3.9. *Let k, $s \geq 0$ be powers of two. Then*

- $n_s(k) \geq \left(1 + \frac{s}{k}\left(\frac{s}{k} + 3\right) + \frac{1}{4}\right)^{\frac{k}{2}} - 1$ *if $k \leq s$;*
- $n_s(k) \geq \left(1 + \frac{8}{7}\left(\frac{k}{s}\right)^2\right)^{\frac{s}{2}} - 1$ *if $k \geq s$.*

Proof. For $k \geq s$, one can stop the recursion with 2 denominations. Indeed, $n_s(2) = (s^2 + 6s + 1)/4 = \frac{s}{2}(\frac{s}{2} + 3) + \frac{1}{4}$ (see for instance [1] and references therein). We then obtain as minimal case $n_{\frac{2s}{k}}(2) = \frac{s}{k}(\frac{s}{k} + 3) + \frac{1}{4}$. Combined with Lemma 3.8, this results in the improved bound

$$n_s(k) \geq \left(1 + \frac{s}{k}\left(\frac{s}{k} + 3\right) + \frac{1}{4}\right)^{\frac{k}{2}} - 1. \tag{3.6}$$

[2]This can give rise to a natural dynamic programming version in order to determine the best cuts of k and s into $k = k_1 + k_2$ and $s = s_1 + s_2$, but then the algorithm is not polynomial-time anymore.

For $k \leq s$, one can stop the recursion with 2 places. Indeed, $n_2(k) \geq \frac{2}{7}k^2$ [21]. Thus, $n_2(\frac{2k}{s}) \geq \frac{8}{7}\left(\frac{k}{s}\right)^2$ and, combining this again with Lemma 3.8, we obtain

$$n_s(k) \geq \left(1 + \frac{8}{7}\left(\frac{k}{s}\right)^2\right)^{\frac{s}{2}} - 1. \tag{3.7}$$ □

Remark 3.10. A possible basis is a geometric progression $\{1, r, \ldots, r^{k-1}\}$, as in [28, Eq. (26)]. In this case, the best common ratio is $r \approx 1 + \frac{s+1}{k}$, which yields a range never exceeding $\left(1 + \frac{s+1}{k}\right)^k - 2$ [28, 29, 30]. If $k = 2$, this is not better than $(s^2 + 6s + 1)/4$, and if $k \geq 3$, this is also always less interesting than the recursive algorithm with Proposition 3.9.

We now turn to the general case, with k and s not powers of two. Then, the following Lemma 3.11, shows that asymptotically, the best lower bound is given by any cut with the same fraction. In particular, the midpoint construction of Lemma 3.8 thus also provides the best lower bound when $k \leq s$.

Lemma 3.11. *Let $g(k, s) = (1 + s/k)^k - 1$ and $h_{k,s}(t, u) = (g(t, u) + 1)(g(k - t, s - u) + 1) - 1$ with $t \in \{1..k - 1\}$ and $u \in \{1..s - 1\}$. Then $g(k, s)$ is a local maximum of $h_{k,s}$.*

Proof. The first derivative of g in u is $g_1 = -t^{-t}(t + u)^{t-1}(k - t)^{t-k}(k - t + s - u)^{k-t-1}(ku - st)$ whose zeroes are $k - t + s$ and st/k. With $t \in \{1..k - 1\}$ and $u \in \{1..s - 1\}$, the former is too large. Now the second derivative of g at $u = st/k$ is $g_2 = -\frac{1}{t}k^{3-k}(k + s)^{t-2}(k - t)^{t-k}((k - t)(k + s))^{k-t-2}(k + s)^2(k - t)$ and is thus negative when $t \in \{1..k - 1\}$. Therefore, $u^* = st/k$ is local maximum of $h_{k,s}$.

Finally, let $v = k/t$ so that $u^* = st/k = s/v$. Then, $h_{k,s}(k/v, s/v) = (g(\frac{k}{v}, \frac{s}{v}) + 1)(g(k\frac{v-1}{v}, s\frac{v-1}{v}) + 1) - 1 = (1 + s/k)^{k/v}(1 + s\frac{v-1}{v}(k\frac{v-1}{v})^{-1})^{k\frac{v-1}{v}} - 1 = (1 + s/k)^{k/v+k-k/v} - 1 = g(k, s)$, independently of the factor v, and the lemma is proven. □

This analysis is of course quite rough for small values of k and s, where the involved constants also matter. For very small values, extremal bases and the extremal s-range are known, see e.g. [5], and can thus be fed to the recursive algorithm before reaching $k = 1$ or $s = 1$, as was shown for instance with Equations (3.6) and (3.7).

3.3. **Hybrid algorithm in practice.** We can summarize the results of Sections 3.1 and 3.2 in terms of asymptotics, in Table 2. In practice, one use a hybrid algorithm based on the different lower bounds, switching to the best regime depending on the values of k and s.

TABLE 2. Asymptotic regimes of the approximation algorithms for different k and s

$k \leq s$	$k \geq s$
Fibonacci: $(s - k)(1 + \varphi)^k$	Balanced: $\left(1 + \frac{k}{s}\right)^s$
D&C: $\left(1 + \frac{s}{k}\left(\frac{s}{k} + 3\right) + \frac{1}{4}\right)^{\frac{k}{2}}$	D&C: $\left(1 + \frac{8}{7}\left(\frac{k}{s}\right)^2\right)^{\frac{s}{2}}$

With good base cases, the recursive algorithm is asymptotically better than Fibonacci, even with $k = s$. For instance indeed, the best known sequence for $k = s = 5$ is $\{1, 4, 9, 31, 51\}$ with s-range 126 (while Fibonacci gives $\{1, 3, 8, 21, 55\}$ with s-range 88). On the one hand, the recursive algorithm with $k = s = 5 \cdot 2^i$, stopping the recursion at threshold 5, shows that $n_s(k) \geq (1 + 126)^{2^i} \approx 2.635^{5 \cdot 2^i}$, from Lemma 3.8. On the other hand, from Corollary 3.4, Fibonacci sequence will give a range closer to $(1 + \varphi)^{5 \cdot 2^i} \approx 2.618^{5 \cdot 2^i}$. This is also shown in practice in Figure 5, where we compare the recursive algorithm and the Fibonacci sequence, using the programs `./bin/basis` and `./bin/fibo` of the GStamps library.

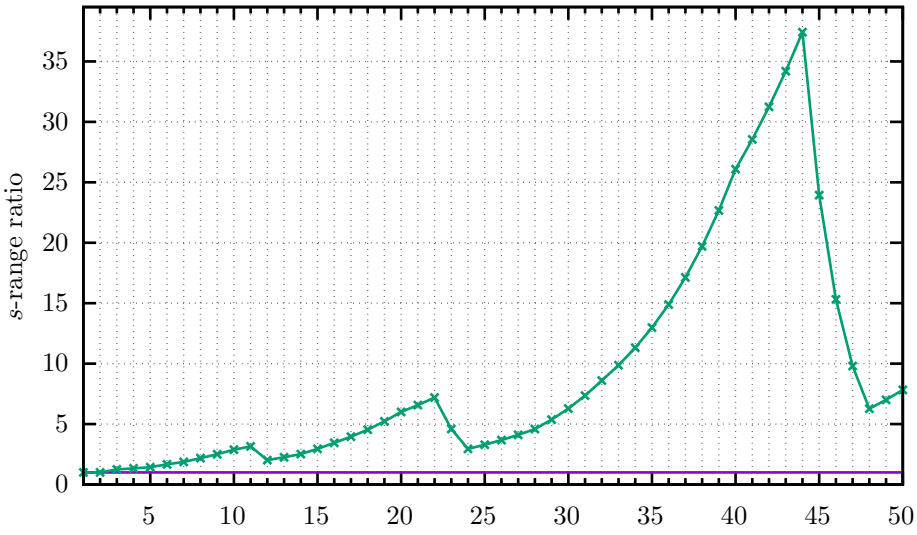

FIGURE 5. Range ratio between the recursive algorithm and the Fibonacci sequence, for $k = s$ (for $k = s \geq 25$ only a lower bound on the range of the basis obtained by the recursive algorithm is used to compute the ratio)

The recursive algorithm with good base cases performs also better than the balanced basis. As presented in Section 3.1, the construction gives a common ratio of $1+\varphi$ when $k = s$ and not much better than $q = k/s$ otherwise, namely $q(1+\epsilon_q)$. The dominant term of the lower bound is thus $(1+q(1+\epsilon_q))^s = q^s(1+\epsilon_q+\frac{1}{q})^s$, with $1+\epsilon_q+\frac{1}{q} = 1+\frac{1}{2}(\sqrt{1+4/q}-1)+\frac{1}{q}$, asymptotically close to $1+\frac{1}{2}(1+2/q-1)+\frac{1}{q} = 1+\frac{2}{q}$. This is to be compared to Equation (3.7), $(1+\frac{8}{7}q^2)^{s/2} = q^s\sqrt{8/7+1/q^2}^s$, where $\sqrt{8/7+1/q^2} = \sqrt{8/7}\sqrt{1+7/(8q^2)}$ is asymptotically close to $\sqrt{8/7}+\sqrt{7/8}\frac{1}{2q^2} > 1$.

This implies that the recursive algorithm with threshold is thus asymptotically better. It is shown in Figure 6 that it is also better for not so large values of $q = k/s$, see the curve "Mid-cut" (except for the point $k = 26, s = 8$: even if the extremal bases for $k = 12, s = 4$ have s-range 700 [5, Addendum], our initial implementation only finds a s-range of 566 at $k = 13, s = 4$). In this figure, "Mid-cut" is obtained with the recursive algorithm with threshold, always dividing at the middle point $k/2, s/2$. In the second curve, "First-cut", we select the best cutting

point, for the first recursive call, with dynamic programming, among $1..k-1, 1..s-1$. It then recursively further divide always at the middle point. Then "Best-cuts" uses a dynamic programming selection at each recursive call. Finally, the curve "Good-basis", incorporates, as base cases to the midpoint construction of "Mid-cut", all the good basis obtained in a programming contests.[3] There, a basis at $k = 13, 4$ has s-ranges up to 878, improving, e.g., the result for $k = 26, s = 8$, to a range of 772647. Now, again, there is a drop at $k = 62, s = 8$, even for this "Good-basis" curve: this is now because the brute force searches of the contest did provide good bases only until $k = 30, s = 4$. Therefore, after $k = 31$ we have a supplementary recursive call which starts to reduce the advantage obtained with better base cases. Note that "Mid-cut" and "Good-basis" represent polynomial-time algorithms, implemented in `./bin/basis`, while "First-cut" and "Best-cuts", both implemented in `./bin/dynprg` with different parameters, are not polynomial-time.

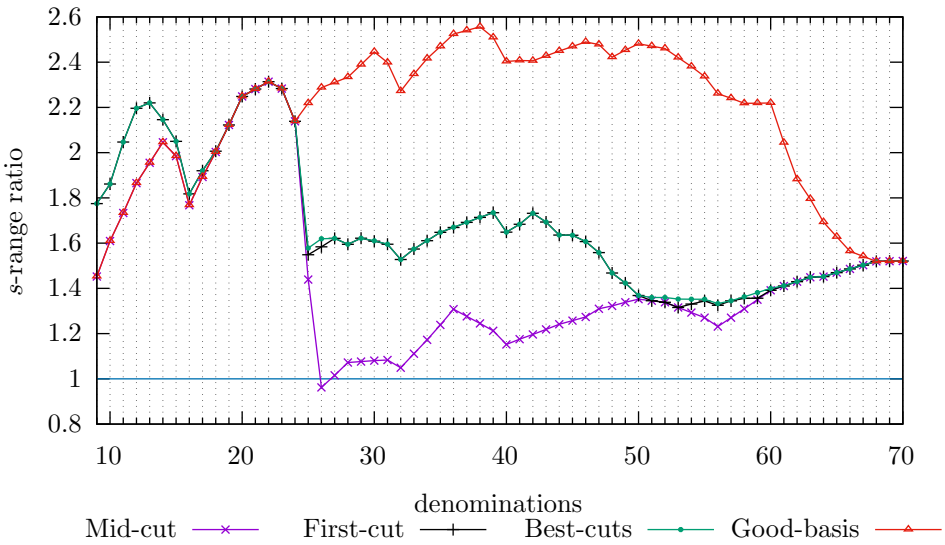

FIGURE 6. s-range ratio, Recursive with threshold divided by that of Corollary 3.5, for $s = 8$

4. Applications to privacy-preserving cryptographic protocols

Private set operation protocols are cryptographic protocols in which two (or more) parties aim to perform some set operations while preserving privacy. For instance in a private set union protocol, a sender holds a set X and a receiver holds a set Y, and the goal is for the receiver to learn the union $X \cup Y$, while it should not learn the intersection $X \cap Y$. Many such protocols, for instance for the union [11] or the intersection [6, 9], rely on polynomial evaluation, with polynomials whose roots are the set elements. In general it boils down to a client owning some value x in a field $\mathbb{F}$, and a server knowing a degree-d polynomial $P \in \mathbb{F}[X]$; the goal being

[3] Al Zimmermann's "Son Of Darts" http://azspcs.com/Contest/SonOfDarts/FinalReport

for the client to learn $P(x)$ while the server does not learn x and the client does not learn the polynomial P.

A classical solution for this problem is for the client to encrypt x using a fully homomorphic encryption scheme (FHE)[4] and then to send it to the server. The latter evaluates its polynomial P on the encryption of x using the homomorphic properties of the encryption scheme. It obtains an encryption of $P(x)$ that the client can then decrypt to obtain $P(x)$. A limitation of this solution is the cost of homomorphic operations. In particular, the efficiency of homomorphic operations is directly related to the *multiplicative depth* of the function to evaluate. The evaluation of a degree-d polynomial requires a multiplicative depth $\log(d)$ (using a divide & conquer evaluation scheme), which soon becomes the bottleneck in practice.

4.1. **Depth trade-off in fully homomorphic encryption.** A fully homomorphic encryption (FHE) scheme is a semantically secure public-key encryption scheme that allows to perform arithmetic operation on ciphertexts resulting, after decryption, in operations on the underlying plaintexts. Namely, a FHE is built with a *key generation* algorithm, generating private and public keys, an *encryption* and a *decryption* algorithms ; the former takes a plaintext and the public key to output a ciphertext, while the latter inputs a ciphertext and the secret key to output a plaintext. Basically, the scheme allows to perform *homomorphic addition*, *plaintext-ciphertext product* and *homomorphic product* on ciphertexts, whose result after decryption to the corresponding operations on plaintexts.

Many of the existing FHE schemes[5] security rely on the difficulty of the "Learning wtih Errors" game: it basically means that a ciphertext is masked with a noise. Each homomorphic operation increases the size of this noise and the decryption remains correct as long as this noise does not exceed a threshold. There are thus extra procedures to control this noise growth.

Now, the most expensive operation in terms of noise expansion is the homomorphic product ; for that reason, the effort is usually essentially put on reducing the amount of homomorphic products, in particular when performed in sequence. In the ring FHE schemes (BGV, BFV, CKKS), each homomorphic product is followed by a *modulus switching* procedure ; the idea is to reduce the size of the ciphertext space in order to reduce the size of the noise. This can be done a finite number of times, let say L times, as long as the ciphertext remains secure. In theory, this L can be as large as needed, as long as the fresh ciphertext space is large enough, for the security to hold. In practice, setting up a context allowing 10 modulus switching, for instance, already implies large computational cost overheads, and a huge communication volume to exchange ciphertexts, and it can possibly threaten the security of the scheme. We remark that the amount of modulus switching usually corresponds to the multiplicative depth of the circuit.

From now on, we consider that L is the *maximum multiplicative depth* allowed by the scheme. Figure 7 shows some experimental results obtained with an Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz on a single thread using the BFV scheme from the openFHE library[6]. We generated contexts for various L, while keeping the same plaintext space ($\mathbb{F}_p$ in our case, with p on 48 bits), and with a security level beyond

[4]Details on fully homomorphic encryption schemes are presented in Section 4.1.

[5]TFHE [8], FHEW[10], CKKS [7], BFV [3], BGV [2].

[6]https://github.com/openfheorg/openfhe-development

128 bits. Figure 7 shows that both the size of a fresh ciphertext and the runtime of one homomorphic product on fresh ciphertexts[7] grow with the maximum depth allowed by the scheme.

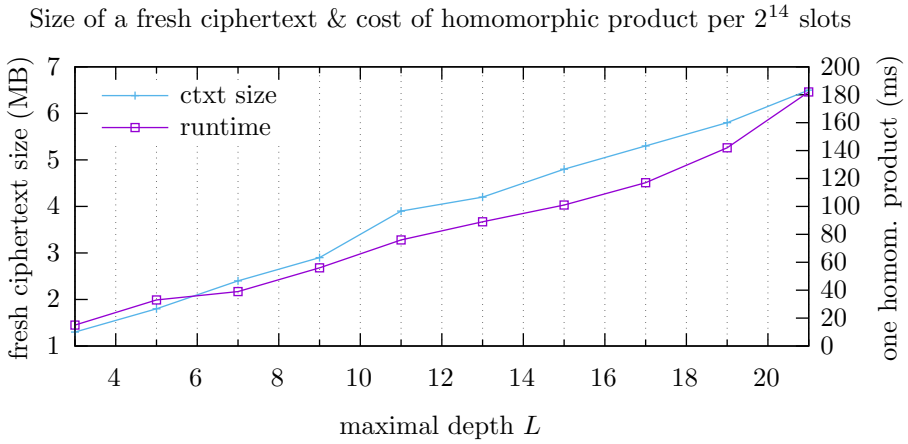

FIGURE 7. Experimental results for various maximum depth using openFHE

Remark 4.1. In practice, one ring FHE ciphertext is large enough to encrypt many plaintexts through batching. We denote that as the number of *slots* of the ciphertext. Performing a homomorphic operation on a ciphertext is actually acting simultaneously on all the slots. The different contexts may have various number of slots depending on the chosen L, due to the fact that a certain security level is imposed. Namely, with the chosen prime and security level, the BFV contexts with $L \in [2, 5]$ have 2^{14} slots, for $L \in [6, 11]$ there are 2^{15} slots, and for larger L (up to 21 in our experiments), we have 2^{16} slots. For consistency of the experiments, all the experimental runtimes are given *per* 2^{14} *slots*: that is the runtimes presented in Figure 7 are the actual runtime for $L \in [2, 5]$ and the actual runtime divided by 2 (resp. divided by 4), for $L \in [6, 11]$ and 2^{15} slots (resp. for $L \geq 12$ and 2^{16} slots). It is also important to note that if not all the slots are used, then this can however not be used to accelerate the runtime for $L \in [6, 11]$ nor for $L \geq 12$.

Once the *modulus switching* procedure is no longer doable, or after each homomorphic products for the FHE schemes on the Torus (FHEW, TFHE), an additional *bootstrapping* procedure can be performed [10]; the idea is to perform a *homomorphic* decryption which resets the noise to (almost) the fresh level. This procedure is extremely expansive in the ring schemes and requires a specific choice of parameters ; in practice, it is often better to try to avoid it. For all these reasons, we see how reducing or at least controlling the multiplicative depth, does matter in practice, especially for the ring schemes that we are focusing on.

[7]The timings are obtained as a mean of 100 products.

4.2. **Multiplicative depth reduction with a stamp basis.** To reduce the multiplicative depth of polynomial evaluation, the idea is for the client to send not only its (encrypted) value x, but also some of its (encrypted) powers x^{a_i}, $i \in [k]$. Viewing $A_k = \{a_1 < \cdots < a_k\}$ as a stamp basis, $n_{2^\ell}(A_k) \geq d$ implies that the server can evaluate any degree-d polynomial P with a multiplicative depth $\ell + 1$. Indeed, this means by definition that any power x^d is a product of at most 2^ℓ powers x^{a_i}, which can be computed in depth ℓ using a binary tree. The extra depth allows for the final inner product of the coefficients of P by the powers of x. As an example, if the client sends x, x^5 and x^8, the server can evaluate any degree-26 polynomial in depth 3 since $n_4(\{1, 5, 8\}) = 26$. For instance, $x^{26} = (x^5 \cdot x^5) \cdot (x^8 \cdot x^8)$, $x^{25} = (x^1 \cdot x^8) \cdot (x^8 \cdot x^8)$, etc.

This idea leads to a trade-off between communication (since several powers are sent) and multiplicative depth, hence efficiency in practice. The trade-off is actually even better than it may seem at first sight. Indeed, reducing the multiplicative depth also reduces the size of each ciphertext. Instead of sending one large ciphertext, the sender sends k smaller ciphertexts. The communication still increases, but by a factor less than k. As far as we know, the explicit use of a stamp basis was first suggested by [9] to build an efficient private set intersection protocol ; although no study of the practical efficiency of this approach was provided. We provide here such an analysis that confirms the relevance of this approach. We use our library GStamps to find small bases allowing to evaluate a polynomial with a fixed multiplicative depth L for various degrees d as shown in Table 3. We denote those constructions as **base-L** for the use of a basis allowing to execute the protocol in depth L. As an example, if we want to perform a homomorphic polynomial evaluation of degree $d = 2^{20}$ under a maximum depth $L = 7$, we use our algorithm `./bin/search` with inputs $s = 2^{L-1}$ and $N = d$. It returns the 5-stamps basis $(1, 52, 705, 13100, 99644)$ which has a 64-range of $1782370 \geq 2^{20}$.

TABLE 3. k = #denominations for basis obtained with `./bin/search`

Constr.	s	k for $d = 2^{10}$	$d = 2^{12}$	$d = 2^{14}$	$d = 2^{16}$	$d = 2^{18}$	$d = 2^{20}$
base-5	16	4	5	6	9	10	12
base-6	32	3	4	5	5	7	8
base-7	64	2	3	4	4	5	**5**

We now compare several protocols for polynomial evaluation using different communication-depth trade-offs in Table 4. The **naive** construction consists in sending only the value x, for a multiplicative depth $\log(d)$. A second construction, following [6, 31], reduces the depth to $O(\log \log d)$ by sending encryptions of x^{2^i}, $i \in [\log d]$. In fact, this can be viewed as a geometric progression stamp basis with common ratio 2 (Remark 3.10). We denote this as the **geom.** construction.

We compare the communication volumes and the expected running times to homomorphically compute all the powers for each construction for various polynomial degrees. The communication volumes presented in Table 4 take into account the number of ciphertexts as well as their sizes. As it happens, not all homomorphic multiplication have the same running time. *Deeper* multiplications are faster. The timing estimations in Table 4 take this aspect into account. The number of multiplications per depth level is explicit for the **naive** and **geom.** constructions and

can be counted, for instance using the program `./bin/depthrange` of GStamps, for any basis, including for our **base-L** constructions[8].

TABLE 4. Communication volumes (MB) and runtime (s) estimation for various polynomial degrees

Constr.	$d = 2^{10}$		$d = 2^{12}$		$d = 2^{14}$		$d = 2^{16}$		$d = 2^{18}$		$d = 2^{20}$	
	MB	s	MB	s	MB	s	MB	s	MB	s	MB	s
naive	**3.7**	36	**4.9**	173	**5.0**	763	**5.5**	3590	**6.1**	16032	**6.8**	69830
geom.	18.3	26	21.9	100	25.5	382	33.9	1745	38.1	6674	42.3	25676
base-5	7.5	21	9.3	**81**	11.1	**311**	16.5	1280	18.3	**4898**	21.9	19375
base-6	6.6	24	8.7	100	10.8	404	10.8	**1190**	15.0	5466	17.0	20683
base-7	5.0	**20**	7.3	98	9.7	504	9.7	1315	12.0	5940	12.0	**18916**

We observe in Table 4 the relevance of the trade-off for the performance. The use of stamps in geometric progression is worse both in communication, as expected, but also in runtime compared to the use of better bases. We can see that a modest increase in communication, with a factor less than two, can result in a large runtime gain, almost a 3.7× speedup. As an example, consider the degree $d = 2^{20}$. To allow a depth 21 as required in the naive construction, one has to send a ciphertext of size 6.55 MB. The answer of the receiver has size 0.25 MB, for a total of 6.8 MB. Using instead the construction **base-7**, the basis has 5 denominations. Since the required depth is only 7, each ciphertext has size 2.35 MB, for a total of 12 MB. To conclude, the use of good stamp bases allows to choose more finely the different parameters of the FHE scheme, and it results in a practical efficiency boost. A good trade off between runtime and communication can, suprisingly, be more efficient both in runtime and communication.

5. CONCLUSION

The local and global postage stamp problems are intriguing. We provide here improved algorithms for both problems, an analysis of their behavior and a software library for dealing with stamp bases. We also show that good bases can be used to accelerate secure multi-party computations on sets. In this latter setting, the Frobenius automorphism in characteristic p, that is $x \hookrightarrow x^{p^n}$, as well as similar rotations, can be homorphically cheap in terms of added noise [12]. Finding bases that contain an important number of such rotations, would then be interesting to further improve secure multi-party computations. More generally, finding good bases with prescribed elements seems to be challenging.

[8]For example, with the basis $(1, 52, 705, 13100, 99644)$, five elements require no depth, fifteen ones require a depth of 1, 105 a depth of 2, 1161 a depth of 3, 19 062 a depth of 4, 393 157 a depth of 5, and the rest need a depth of 6

References

[1] Ronald Alter and Jeffrey A Barnett, *Remarks on the postage stamp problem with applications to computers*, Congressus Numerantium 19, Proc. Eighth Southeastern Conference on Combinatorics, Graph Theory, and Computing (Baton Rouge, La.), 1977, pp. 43–59, available from https://notatt.com/lsu-stamp.pdf.

[2] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan, *(Leveled) fully homomorphic encryption without bootstrapping*, ACM Trans. Comput. Theory **6** (2014), no. 3, 13:1–13:36, doi:10.1145/2633600.

[3] Zvika Brakerski and Vinod Vaikuntanathan, *Fully homomorphic encryption from ring-LWE and security for key dependent messages*, Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings (Phillip Rogaway, ed.), Lecture Notes in Computer Science, vol. 6841, Springer, 2011, pp. 505–524, doi:10.1007/978-3-642-22792-9_29.

[4] M. F. Challis, *Two New Techniques for Computing Extremal h-bases A_k*, The Computer Journal **36** (1993), no. 2, 117–126, doi:10.1093/comjnl/36.2.117.

[5] Michael F Challis and John P Robinson, *Some Extremal Postage Stamp Bases*, Journal of Integer Sequences **13** (2010), no. 10.2.3, 15, available from https://cs.uwaterloo.ca/journals/JIS/VOL13/Challis/challis6.html.

[6] Hao Chen, Kim Laine, and Peter Rindal, *Fast private set intersection from homomorphic encryption*, Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017 (Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, eds.), ACM, 2017, pp. 1243–1255, doi:10.1145/3133956.3134061.

[7] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song, *Homomorphic encryption for arithmetic of approximate numbers*, Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I (Tsuyoshi Takagi and Thomas Peyrin, eds.), Lecture Notes in Computer Science, vol. 10624, Springer, 2017, pp. 409–437, doi:10.1007/978-3-319-70694-8_15.

[8] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène, *TFHE: fast fully homomorphic encryption over the torus*, J. Cryptol. **33** (2020), no. 1, 34–91, doi:10.1007/S00145-019-09319-X.

[9] Kelong Cong, Radames Cruz Moreno, Mariana Botelho Da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg, *Labeled PSI from Homomorphic Encryption with Reduced Computation and Communication*, Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (Virtual Event Republic of Korea), ACM, November 2021, pp. 1135–1150, doi:10.1145/3460120.3484760.

[10] Léo Ducas and Daniele Micciancio, *FHEW: bootstrapping homomorphic encryption in less than a second*, Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I (Elisabeth Oswald and Marc Fischlin, eds.), Lecture Notes in Computer Science, vol. 9056, Springer, 2015, pp. 617–640, doi:10.1007/978-3-662-46800-5_24.

[11] Jean-Guillaume Dumas, Alexis Galan, Bruno Grenet, Aude Maignan, and Daniel S. Roche, *Optimal communication unbalanced private set union*, Applied Cryptography and Network Security - 23rd International Conference, ACNS 2025, Munich, Germany, June 23-26, 2025, Proceedings, Part II (Marc Fischlin and Veelasha Moonsamy, eds.), Lecture Notes in Computer Science, vol. 15826, Springer, 2025, pp. 107–135, doi:10.1007/978-3-031-95764-2_5.

[12] Shai Halevi and Victor Shoup, *Algorithms in HElib*, Advances in Cryptology – CRYPTO 2014 (Berlin, Heidelberg) (Juan A. Garay and Rosario Gennaro, eds.), Springer Berlin Heidelberg, 2014, pp. 554–571, doi:10.1007/978-3-662-44371-2_31.

[13] Christoph Kirfel, *On Extremal Bases for the h-range Problem, I*, Tech. Report 53, Department of Mathematics, University of Bergen, 1989, available from https://bora.uib.no/bora-xmlui/bitstream/handle/1956/19569/On-extremal-bases-for-the-h-range-problem-I.pdf.

[14] ———, *On Extremal Bases for the h-range Problem, II*, Tech. Report 55, Department of Mathematics, University of Bergen, 1990, available from https://bora.uib.no/bora-xmlui/bitstream/handle/1956/19570/On-extremal-bases-for-the-h-range-problem-II.pdf.
[15] William Frederick Lunnon, *A postage stamp problem*, The Computer Journal **12** (1969), no. 4, 377–380, doi:10.1093/comjnl/12.4.377.
[16] Chris Mitchell, *Another Postage Stamp Problem*, The Computer Journal **32** (1989), no. 4, 374–376, doi:10.1093/comjnl/32.4.374.
[17] Svein Mossige, *Algorithms for Computing the h-Range of the Postage Stamp Problem*, Mathematics of Computation **36** (1981), no. 154, 575–582, doi:10.1090/S0025-5718-1981-0606515-1.
[18] ———, *The postage stamp problem: an algorithm to determine the h-range on the h-range formula on the extremal basis problem for $k = 4$*, Mathematics of Computation **69** (1999), no. 229, 325–337, doi:10.1090/S0025-5718-99-01204-1.
[19] ———, *The Postage Stamp Problem: The Extremal Basis for k=4*, Journal of Number Theory **90** (2001), no. 1, 44–61, doi:10.1006/jnth.2000.2620.
[20] Arnulf Mrose, *Ein rekursives Konstruktionsverfahren für Abschnittsbasen.*, Journal für die reine und angewandte Mathematik (Crelles Journal) **1974** (1974), no. 271, 214–217, doi:10.1515/crll.1974.271.214.
[21] ———, *Untere Schranken für die Reichweiten von Extremalbasen fester Ordnung*, Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg **48** (1979), no. 1, 118–124, doi:10.1007/BF02941296.
[22] Hans Rohrbach, *Ein beitrag zur additiven zahlentheorie*, Mathematische Zeitschrift **42** (1937), no. 1, 1–30, available from https://eudml.org/doc/168701.
[23] Ernst S Selmer, *On the postage stamp problem with three stamp denominations; I*, Mathematica Scandinavica **47** (1980), 29–71, doi:10.7146/math.scand.a-11874.
[24] ———, *On the postage stamp problem with three stamp denominations, III*, Mathematica Scandinavica **56** (1985), 105–116, doi:10.7146/math.scand.a-12091.
[25] ———, *Associate bases in the postage stamp problem*, Journal of Number Theory **42** (1992), no. 3, 320–336, doi:10.1016/0022-314X(92)90097-9.
[26] Ernst S Selmer and Arne Rôdne, *On the postage stamp problem with three stamp denominations, II*, Mathematica Scandinavica **53** (1983), 145–156, doi:10.7146/math.scand.a-12022.
[27] Jeffrey Shallit, *The computational complexity of the local postage stamp problem*, ACM SIGACT News **33** (2002), no. 1, 90–94, doi:10.1145/507457.507473.
[28] Alfred Stöhr, *Gelöste und ungelöste fragen über basen der natürlichen zahlenreihe. i.*, Journal für die reine und angewandte Mathematik **1955** (1955), no. 194, 40–65, doi:doi:10.1515/crll.1955.194.40.
[29] ———, *Gelöste und ungelöste fragen über basen der natürlichen zahlenreihe. ii.*, Journal für die reine und angewandte Mathematik **1955** (1955), no. 194, 111–140, doi:doi:10.1515/crll.1955.194.111.
[30] Amitabha Tripathi, *The postage stamp problem and arithmetic in base r*, Czechoslovak Mathematical Journal **58** (2008), no. 4, 1097–1100, doi:10.1007/s10587-008-0071-2.
[31] Binbin Tu, Yu Chen, Qi Liu, and Cong Zhang, *Fast unbalanced private set union from fully homomorphic encryption*, Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023 (Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, eds.), ACM, 2023, pp. 2959–2973, doi:10.1145/3576915.3623064.
[32] Gang Yu, *A new upper bound for finite additive h-bases*, Journal of Number Theory **156** (2015), 95–104, doi:10.1016/j.jnt.2015.04.007.

Univ. Grenoble Alpes, Laboratoire Jean Kuntzmann, UMR CNRS 5224, 150 place du Torrent, IMAG - CS 40700, 38058 Grenoble cedex 9, France
Email address: {Firstname.Lastname}@univ-grenoble-alpes.fr