Algorithms for the local and the global postage stamp problem

Léo Colisson Palais
Univ. Grenoble Alpes, Laboratoire Jean Kuntzmann, UMR CNRS 5224, 150 place du Torrent, IMAG - CS 40700, 38058 Grenoble cedex 9, France

Jean-Guillaume Dumas
Univ. Grenoble Alpes, Laboratoire Jean Kuntzmann, UMR CNRS 5224, 150 place du Torrent, IMAG - CS 40700, 38058 Grenoble cedex 9, France

Alexis Galan
Univ. Grenoble Alpes, Laboratoire Jean Kuntzmann, UMR CNRS 5224, 150 place du Torrent, IMAG - CS 40700, 38058 Grenoble cedex 9, France

Bruno Grenet
Univ. Grenoble Alpes, Laboratoire Jean Kuntzmann, UMR CNRS 5224, 150 place du Torrent, IMAG - CS 40700, 38058 Grenoble cedex 9, France

Aude Maignan
Univ. Grenoble Alpes, Laboratoire Jean Kuntzmann, UMR CNRS 5224, 150 place du Torrent, IMAG - CS 40700, 38058 Grenoble cedex 9, France

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.

2012 ACM Subject Classification Computing methodologies → Number theory algorithms

Keywords and phrases Postage stamp problem, Secure multi-party computation

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 $A_k = \{a_1, a_2, \ldots, a_k\}$ and at most s places to 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.

▶ **Definition 1.** *For a given positive integers 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 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 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 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 [21] 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 [21, 14, 20, 22, 25, 23, 15, 12, 13, 24, 4, 18, 31]. For instance, [5], provides a survey of known extremal s-range and extremal bases for small values of k and s. Then [26] 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 the both the local and the global problems. We first survey the different known methods that can compute "good" basis for the global problem : this means that for given a number of denominations k and positions s, we obtain a basis of denominations with an s-range with the same asymptotic than 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.

Then, we also 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 [16], we were able, first, to remove an asymptotic dependency in s, and, second, to also remove an s factor in the required memory. 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 homomomorphic encryption.

Paper organization. Section 2 presents 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. Then, in Section 3, we give two novel algorithms for the local problem that improve on the time complexity bound and on the amount of required memory. 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 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 $n_s(A_{k,s})$.

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

We review some existing constructions. First we focus on a construction of Alter and Barnett [1]. They provide, for any $k \geq s$, a pretty 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 the construction in the case $k \geq s$. Although they provide the correct value for $n_s(A_{k,s})$, their proof only prove the lower bound. We provide a full proof of the upper bound that applies both to their original construction 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 [19] 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, the database of base cases, we obtain the best known asymptotic constructions as well as experimental validation of these results. Using our GStamps library, one can retrieve all the result presented in this part. Namely, our algorithms *fibo, alba, greedy, geom* compute the s-ranges for (respectively) the Fibonacci, the Alter and Barnett and its variant with respect to Proposition 7, and the geometric basis constructions, for chosen k, s. Our algorithm *basis* computes the best found basis for given k, s using the recursive algorithm.

2.1 Fibonacci sequence

We first describe the Alter and Barnett's construction focus on the case $k = s$. Let $(f_i)_{i\geq 0}$ be the Fibonacci sequence defined by $f_0 = 0$, $f_1 = 1$ and $f_{i+2} = f_i + f_{i+1}$ for $i \geq 0$. The Fibonacci stamps is the set $F_k = \{f_{2i}\}_{i\in[1,k]}$. Alter and Barnett [1] state that $n_k(F_k) = f_{2k+1} - 1$, while they only prove that $n_k(F_k) \geq f_{2k+1} - 1$.

▶ **Proposition 5** (Proof in Appendix A.2). *The k-range of F_k is $n_k(F_k) = f_{2k+1} - 1$.*

Let $\varphi = \varphi_+ = \frac{1+\sqrt{5}}{2} \approx 1.618$ be the golden number, then $\varphi^2 = 1 + \varphi$ and let $\varphi_- = \frac{1-\sqrt{5}}{2} = 1 - \varphi = -\varphi^{-1}$. Then we have that:

$$f_n = \frac{1}{\sqrt{5}}(\varphi_+^n - \varphi_-^n) = \left\lfloor \frac{1}{\sqrt{5}}\varphi^n \right\rceil \tag{1}$$

Now this basis can be used also if $k \leq s$ by just adding integers obtained by groups of k. Corollary 6 shows that the resulting s-range then grows accordingly.

▶ **Corollary 6** (Proof in Appendix A.2). *For $k \leq s$, let $s = qk + r$ by Euclidean division, then:* $\left\lceil \frac{s}{\sqrt{5}} \right\rceil (1+\varphi)^k > n_s(F_k) > \left(\left\lfloor \frac{s}{k} \right\rfloor \left(1 - \frac{\sqrt{5}}{\varphi^{2k+1}}\right) + \frac{1}{\varphi^{2k-2r}} \right) \frac{\varphi}{\sqrt{5}} (1+\varphi)^k$.

2.2 Greedy piece-wise linear algorithm

Alter and Barnett provide a general construction for $k \geq s$. Writing $k = qs + r$ with $r < s$, they defined a basis G_k as the concatenation of s blocks $B_1, \ldots, B_s$ such that $|B_i| = q$ for $i < s$ and $|B_s| = q + r$, and the denomination within each block are in arithmetic progression.

For each $i \in [s]$, let $G'_i = B_1 \| \cdots \| B_i$ and let $U_i = n_i(G'(i))$. They state the following induction relation:

$$\text{[1, Eq. (20)]} \quad U_{t+2} = (q+2)U_{t+1} - U_t + q, U_0 = 0, U_1 = q. \tag{2}$$

They deduce that

$$n_s(G_k) = \sum_{i=1}^{s} \binom{s+i}{2i} q^i + r \sum_{i=0}^{s-1} \binom{s+i-1}{2i} q^i. \tag{3}$$

As mentioned already, although this value is correct, they only prove that $n_s(G_k)$ is at least the right-hand side of (3). We provide in Appendix A.2 a full proof of the upper bound.

Solving exactly Equation (2), one gets that the dominant term of Equation (3) is in fact: $\left[\frac{1}{2}\left(1-\sqrt{1-\frac{4}{q+4}}\right)+\frac{q}{\sqrt{q}\sqrt{q+4}}\right]\left(1+q\frac{1+\sqrt{1+\frac{4}{q}}}{2}\right)^s$. The common ratio boils down to $1+\varphi$ when $q=1$. Otherwise, the ratio is asymptotically close to $1+q=1+\frac{k}{s}$. With $\epsilon_q=\frac{\sqrt{1+4/q}-1}{2}$, we thus obtain a basis whose s-range is asymptotically

$$\alpha\left(1+\frac{k}{s}(1+\epsilon_q)\right)^s. \tag{4}$$

Actually, we can improve Alter and Barnett's construction. The idea is to better balance the blocks in G_k. Instead of using $s-1$ blocks of size q and one of size $q+r$, one can use $(s-r)$ blocks of size q and r of size $q+1$, with the same construction of the blocks. The construction provides a larger asymptotic for $n_s(G_k)$.

▶ **Proposition 7** (Proof in Appendix A.2). *Using the variant of Alter and Barnett's construction with r blocks of size $(q+1)$ and then $(s-r)$ blocks of size q, the dominant term of $n_s(G_k)$ becomes $(1+(q+1)(1+\epsilon_{q+1}))^r\,(1+q(1+\epsilon_q))^{s-r}$.*

2.3 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 [19, 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 8.

▶ **Proposition 8** (Proof in Appendix A.2). $n_{s_1+s_2}(k_1+k_2)\geq(n_{s_1}(k_1)+1)(n_{s_2}(k_2)+1)-1$

The constructive proof of Proposition 8 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 asymptotic, with the help of a simpler function in Lemma 9. This will provide 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 9** (Proof in Appendix A.2). *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 $f(i)=(1+T)^{2^i}-1$.*

Now, suppose that $2^{i+j}=k\geq s=2^i$. We use a *midpoint construction*, cutting both parameters in halves. The recusive cuts then stops with the minimal case such that $n_1(\frac{k}{s})=\frac{k}{s}$. Thus, letting $T=\frac{k}{s}$ in Lemma 9, we obtain the following lower bound for a recursive Divide & Conquer algorithm cutting in halves:

$$n_s(k)\geq\left(1+\frac{k}{s}\right)^s-1. \tag{5}$$

This is similar to the asymptotics of Equation (4), but slightly less interesting since there the common ratio is $(1+q(1+\epsilon_q))$, indeed slightly better than $1+q=\left(1+\frac{k}{s}\right)$.

[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 the opposite case, namely $2^{i+j} = s \geq k = 2^i$, now the minimal case of the midpoint construction gives $n_{\frac{s}{k}}(1) = \frac{s}{k}$ resulting to the dual lower bound:

$$n_s(k) \geq \left(1 + \frac{s}{k}\right)^k - 1. \tag{6}$$

This is now to be compared with Corollary 6 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 $\frac{s}{k} < \varphi$.

Now the lower bound of Equation (6) can be refined by stopping the recursion earlier, for instance at $k = 2$. 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 9, this results in the following improved bound:

$$n_s(k) \geq \left(1 + \frac{s}{k}\left(\frac{s}{k} + 3\right) + \frac{1}{4}\right)^{\frac{k}{2}} - 1. \tag{7}$$

▶ Remark 10. Note that a possible basis could be in geometric progression $\{1, r, \ldots, r^{k-1}\}$, as in [27, Eq. (26)]. In this case, the best common ratio is $r \approx 1 + \frac{s+1}{k}$, which yields a range never exceeding $\left(1 + \frac{s+1}{k}\right)^k - 2$ [27, 28, 29]. 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 Equation (7).

As with Equations (6) and (7), the lower bound of Equation (5) can be refined by stopping the recursion earlier, for instance at $s = 2$. [20] has indeed shown that $n_2(k) \geq \frac{2}{7}k^2$. Thus, $n_2(\frac{2k}{s}) \geq \frac{8}{7}\left(\frac{k}{s}\right)^2$ and we obtain:

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

We now turn to the general case, with k and s not powers of two. Then, the following Lemma 11, shows that asymptotically, the best lower bound is given by any cut with the same fraction. In particular, the midpoint construction of Lemma 9 thus also provides the best lower bound when $k \leq s$.

▶ **Lemma 11** (Proof in Appendix A.2)**.** *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 $H_{k,s}$.*

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 (7) and (8).

▶ Remark 12. In practice this makes the recursive algorithm 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, has its lower bound reaches at least $(1 + 126)^{2^i} \approx 2.635^{5 \cdot 2^i}$, from Lemma 9.On the other hand, from Corollary 6, Fibonacci sequence will never give a range greater than $(1+\varphi)^{5 \cdot 2^i} \approx 2.618^{5 \cdot 2^i}$. This is also shown in practice in Figure 3.

▶ Remark 13. The recursive algorithm with good base cases performs also better than the Greedy algorithm. We have seen that Equation (4) again 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 (8), $(1+\frac{8}{7}q^2)^{s/2} = q^s\sqrt{8/7+1/q^2}^s$, where $\sqrt{8/7+1/q^2} = \sqrt{8/7}\sqrt{1+7/(8q^2)}$ is asymptotically close to $\sqrt{8/7}+\sqrt{7/8}\frac{1}{2q^2} > 1$.

Overall, using Remarks 12 and 13, we obtain Table 1 that shows the dominant terms of cost of the algorithms, depending on how k and s compare.

Table 1 Asymptotic regimes of the approximation algorithms for different k and s

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

From Remark 13, the recursive algorithm with threshold is thus asymptotically better. It is shown in Figure 1 that it is also better for not so large values of $q = k/s$. Indeed, in Figure 1, this is shown by the curve "Mid-cut" (except for the point $k = 26, s = 8$, since even if the extremal bases for $k = 12, s = 4$ have s-range 700 [5, Addendum], but 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 the result for $k = 26, s = 8$. Now, again, there is a drop at $k = 62, s = 8$, even for this curve "Good-basis": 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 start 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.

3 s-range algorithms for large envelopes

3.1 Mossige s-range

For the sake of completeness we give in Appendix A.3, Algorithm 2, a version of the algorithm of [16] computing the s-range of a basis. 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 added to one of a_1 to a_k are then attainable. We thus have Proposition 14.

▶ **Proposition 14** ([16]). *Algorithm 2 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 $n_s(A_k) \leq sa_k$.

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

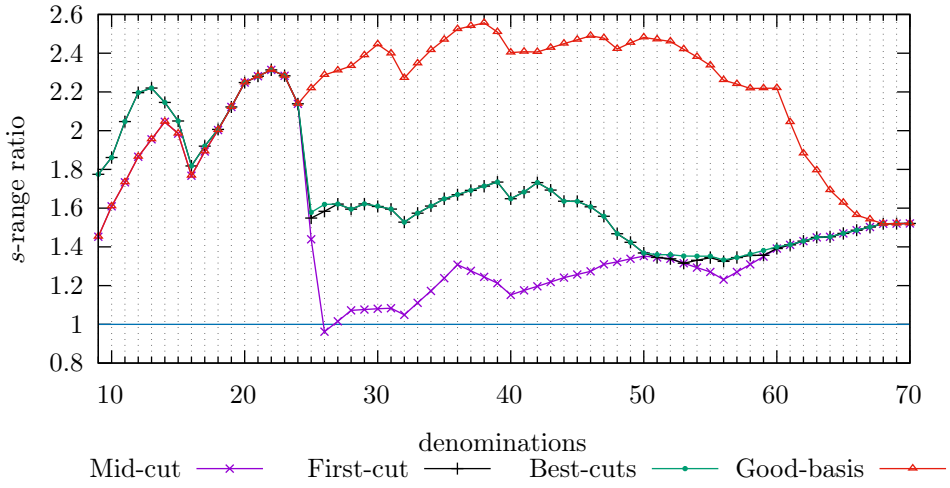

Figure 1 s-range ratio, Recursive with threshold vs Proposition 7, $s = 8$

3.2 Incremental LPSP algorithm

We describe in Algorithm 3 a new algorithm computing the maximum coverage given a fixed set of stamp values $a_1 < \cdots < a_k$ and a maximum number of allowed stamps s. This algorithm outperforms [16] both asymptotically and in practice when s is large enough, and is also arguably simpler. Our new algorithm runs in time $O(ksa_k)$ instead of $O(ks^2a_k)$, and we propose in the next section an improvement to have a better asymptotic time complexity $O(kn)$ with a smaller memory footprint scaling with $O(a_k)$ instead of $O(n)$ that is needed in this first version and in the s-range algorithm [16].

Our first 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 being that we know how to generate the integer i using $T[i]$ stamps when $T[i] \leq s$, while we don't know 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 greatest 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 $i + a_j$, we update its value, i.e. if $T[i + a_j] > T[i] + 1$ we just need to compute $T[i + a_j] \leftarrow T[i] + 1$ and $U[i + a_j] \leftarrow j$. It is easy to see that after this operation, $T[i + 1]$ now contains 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 15) 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 3 in Appendix A.3 for the details).

▶ **Proposition 15** (Proof in Appendix A.3). *There exists an LPSP algorithm which requires $O(ksa_k)$ assignments or more precisely $O(kn_s(A_k) + sa_k)$.*

3.3 Sliding window

We describe in Algorithm 1 an improvement of Algorithm 3 that is more space and time efficient, the size of the memory $O(a_k)$ scaling only with the maximum stamp value a_k, and we also avoid to initialize T to a too large value sa_k approximating n, hence replacing the dependency on sa_k into a dependency on n. This algorithm may also perform better due to better cache optimizations since the table is now smaller and may even entirely fit in the cache.

Algorithm 1 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}
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
  T[i & w] ← s + 1
  i ← i + 1
until T[i & w] > s
return i − 1
```

The improvement of Algorithm 1 over Algorithm 3 comes from the main observation that in Algorithm 3 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 16** (Proof in Appendix A.3)**.** *Algorithm 1 is correct and requires $O(ksa_k)$ assignments or more precisely $O(kn_s(k) + a_k)$.*

3.4 Comparison to the state of the art

Table 2 compares the asymptotical complexity of our two algorithms (Algorithms 1 and 3) with [16].

We also compare those three algorithms based on numerical simulations to also assert the practicality of our approach. [16] remains faster only for very small values of s (smaller than 5 in our simulations) as shown in Figure 4 for $s = 5$ and $s = 10$, or Figures 5 and 6 for various s. Note that a the early termination condition of Selmer's Lemma (see [22, 25, 23]) can be added to all the presented range algorithms : when the condition of the Lemma is

Table 2 Comparison of asymptotic complexity

	[16]	Algorithm 3	Algorithm 1
Time	$O(ks^2a_k)$	$O(ksa_k)$	$O(ksa_k)$
Memory	$O(sa_k)$	$O(sa_k)$	$O(a_k)$

met during the computation, the range is then known [17, (10)]. For instance, Figure 5 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.

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 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 : The client owns some value x in $\mathbb{F}$ and the server a degree-d polynomial $P \in \mathbb{F}[X]$, and the goal is for the client to learn $P(x)$ while neither does the server learns x nor the receiver P.

An easy solution for this problem is for the client to encrypt x using a fully homomorphic encryption scheme (FHE)[4] and send it to the server. The later 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 $\log(d)$, which becomes soon the bottleneck in practice.

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 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 $n_4(\{1, 5, 8\}) = 26$. Indeed, $x^{26} = (x^5 \cdot x^5) \cdot (x^8 \cdot x^8)$.

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 on the practical efficiency of this approach was provided. We provide 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

[4] Details on fully homomorphic encryption schemes are presented in Appendix A.1.

constructions as **base-L** for the use of a basis allowing to do the protocol in depth L. As an example, if we want to perform the protocol with $d = 2^{20}$ and under a maximum depth $L = 7$, we use our algorithm `./bin/search` with inputs $s = 2^{L-1}$ and $N = d$. It returns the stamp basis $(1, 52, 705, 13100, 99644)$ which has a s-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 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, 30], reduces the depth to $O(\log \log d)$ by sending sends 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 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. Although it is easy to exactly count the number of multiplications per depth for the **naive** and **geom.** constructions, we rely on our program `./bin/depthrange` from our library for the **base-L** constructions.

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 huge 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.

References

1 Ronald Alter and Jeffrey A Barnett. Remarks on the postage stamp problem with applications to computers. *Graph Theory*, 1977. URL: 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(3):13:1–13:36, 2014. doi:10.1145/2633600.

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

4 M. F. Challis. Two New Techniques for Computing Extremal h-bases Ak. *The Computer Journal*, 36(2):117–126, February 1993. 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(10.2.3), 2010. URL: 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. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1243–1255. ACM, 2017. doi:10.1145/3133956.3134061.

7 Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *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*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017. 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(1):34–91, 2020. 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. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1135–1150, Virtual Event Republic of Korea, November 2021. ACM. doi:10.1145/3460120.3484760.

10 Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *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*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, 2015. 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. In Marc Fischlin and Veelasha Moonsamy, editors, *Applied Cryptography and Network Security - 23rd International Conference, ACNS 2025, Munich, Germany, June 23-26, 2025, Proceedings, Part II*, volume 15826 of *Lecture Notes in Computer Science*, pages 107–135. Springer, 2025. doi:10.1007/978-3-031-95764-2_5.

12 Christoph Kirfel. On Extremal Bases for the h-range Problem, I. Technical Report 53, Department of Mathematics, University of Bergen, 1989. URL: https://bora.uib.no/bora-xmlui/bitstream/handle/1956/19569/On-extremal-bases-for-the-h-range-problem-I.pdf?sequence=1&isAllowed=y.

13 Christoph Kirfel. On Extremal Bases for the h-range Problem, II. Technical Report 55, Department of Mathematics, University of Bergen, 1990. URL: https://bora.uib.no/bora-xmlui/

bitstream/handle/1956/19570/On-extremal-bases-for-the-h-range-problem-II.pdf?sequence=1&isAllowed=y.

14 W. F. Lunnon. A postage stamp problem. *The Computer Journal*, 12(4):377–380, April 1969. doi:10.1093/comjnl/12.4.377.

15 C. Mitchell. Another Postage Stamp Problem. *The Computer Journal*, 32(4):374–376, April 1989. doi:10.1093/comjnl/32.4.374.

16 Svein Mossige. Algorithms for Computing the h-Range of the Postage Stamp Problem. *Mathematics of Computation*, 36(154):575–582, 1981. URL: https://www.jstor.org/stable/pdf/2007661.pdf.

17 Svein Mossige. 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(229):325–337, 1999. doi:10.1090/S0025-5718-99-01204-1.

18 Svein Mossige. The Postage Stamp Problem: The Extremal Basis for k=4. *Journal of Number Theory*, 90(1):44–61, September 2001. doi:10.1006/jnth.2000.2620.

19 Arnulf Mrose. Ein rekursives Konstruktionsverfahren für Abschnittsbasen. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1974(271):214–217, November 1974. doi:10.1515/crll.1974.271.214.

20 Arnulf Mrose. Untere Schranken für die Reichweiten von Extremalbasen fester Ordnung. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 48(1):118–124, April 1979. doi:10.1007/BF02941296.

21 Hans Rohrbach. Ein beitrag zur additiven zahlentheorie. *Mathematische Zeitschrift*, 42(1):1–30, 1937. URL: https://eudml.org/doc/168701.

22 Ernst S Selmer. On the postage stamp problem with three stamp denominations; I. *Mathematica Scandinavica*, 1980. doi:10.7146/math.scand.a-11874.

23 Ernst S Selmer. On the postage stamp problem with three stamp denominations, III. *Mathematica Scandinavica*, 1985. doi:10.7146/math.scand.a-12091.

24 Ernst S Selmer. Associate bases in the postage stamp problem. *Journal of Number Theory*, 42(3):320–336, November 1992. doi:10.1016/0022-314X(92)90097-9.

25 Ernst S Selmer and Arne Rôdne. On the postage stamp problem with three stamp denominations, II. *Mathematica Scandinavica*, 1983. doi:10.7146/math.scand.a-12022.

26 Jeffrey Shallit. The computational complexity of the local postage stamp problem. *ACM SIGACT News*, 33(1):90–94, March 2002. doi:10.1145/507457.507473.

27 Alfred Stöhr. Gelöste und ungelöste fragen über basen der natürlichen zahlenreihe. i. *Journal für die reine und angewandte Mathematik*, 1955(194):40–65, 1955. doi:doi:10.1515/crll.1955.194.40.

28 Alfred Stöhr. Gelöste und ungelöste fragen über basen der natürlichen zahlenreihe. ii. *Journal für die reine und angewandte Mathematik*, 1955(194):111–140, 1955. doi:doi:10.1515/crll.1955.194.111.

29 Amitabha Tripathi. The postage stamp problem and arithmetic in base r. *Czechoslovak Mathematical Journal*, 58(4):1097–1100, December 2008. doi:10.1007/s10587-008-0071-2.

30 Binbin Tu, Yu Chen, Qi Liu, and Cong Zhang. Fast unbalanced private set union from fully homomorphic encryption. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 2959–2973. ACM, 2023. doi:10.1145/3576915.3623064.

31 Gang Yu. A new upper bound for finite additive h-bases. *Journal of Number Theory*, 156:95–104, November 2015. doi:10.1016/j.jnt.2015.04.007.

A Appendix

A.1 Fully homomorphic encryption (FHE)

A fully homomorphic encryption scheme is a semantically secure public-key encryption scheme that allow to perform arithmetic operation on ciphertexts resulting, after decryption, to 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* algorithm ; the first takes a plaintext and the public key to output a ciphertext, while the second 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.

Most of the existing FHE schemes (TFHE [8], FHEW[10], CKKS [7], BFV [3], BGV [2]) security rely on the difficulty of the LWE game ; it basically means that a ciphertext is masked with a noise. Each homomorphic operation increase the size of this noise and the decryption remains correct as long as this size does not exceed a threshold. For that reason there exist some procedures to control the noise growth. The most expensive operation in term of noise expansion is the homomorphic product ; for that reason, the focus is put essentially on reducing the amount of homomorphic products, in particular 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 time, let say L times, as long as the ciphertext remains secure. In theory, this L can be as big as needed, as long as the fresh ciphertext space is big enough. In practice, setting up a context allowing 10 modulus switchings, for example, already implies computational cost overheads, a huge communication volume to exchange ciphertexts, and possibly can threaten the security of the scheme. We remark that the amount of modulus switching corresponds to the multiplicative depth of the circuit. From now on, we consider that L is the *maximum multiplicative depth* allowed by the scheme.

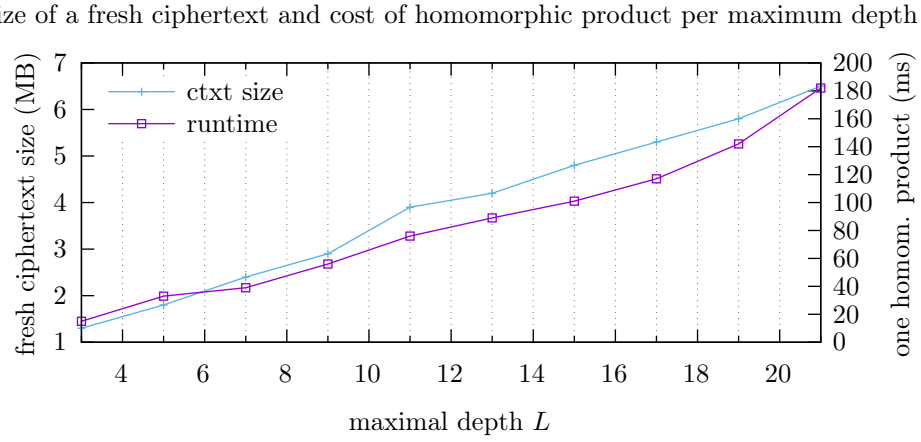

Figure 2 Experimental results for various maximum depth using openFHE

Figure 2 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[5]. We generated contexts for various L, while keeping the same plaintext space, which is $\mathbb{F}_p$ with p on 48 bits, and with a security level beyond 128 bits. Figure 2 shows that both the size of a fresh ciphertext and the runtime of one homomomorphic product on fresh ciphertexts[6] grow with the maximum depth allowed by the scheme.

▶ Remark 17. In practice, one ring FHE ciphertext is big 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 forced to act simultaneously on all the slots. The different contexts may have various number of slots depending on the L chosen, due to the fact that we impose a certain security level. Namely, the contexts with $L \in [2,5]$ have 2^{14} slots, for $L \in [6,11]$ there are 2^{15} slots, and for bigger L (up to 21 in our experiments), we have 2^{16} slots. For consistency of the experiments, all the experimental results given in Section 4 are runtimes and volumes *per* 2^{14} *slots*. For example, the sizes and runtimes for schemes with $L \geq 12$ are, in real life, 4 times bigger that the one in Figure 2, but we can technically do 4 times the job using SIMD. An important remark is that if not all the slots are used, then the results for $L \in [6,11]$ and, even more, for $L \geq 12$ are much worse.

Once the *modulus switching* procedure is no longer doable, or after each homomorphic products for the FHE schemes on the Torus (FHEW, TFHE), the *bootstrapping* procedure can be done [10]; it basically consists in 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 tried to avoid this. For those reasons, we can understand how reducing or atleast controlling the multiplicative depth can matter in practice, especially for the ring schemes that we are focusing on.

A.2 Proofs of Section 2

We first provide the proof of the s-range of Alter and Barnett's construction [1]. We generalize their construction as follows.

Let $q_1, \ldots, q_s \geq 1$. We define a stamp basis as the union of s blocks of size $q_1, \ldots, q_s$. Each block is an arithmetic progression. The first block is $B_1 = \{1, \ldots, q_1\}$. Assume that the first i blocks have been built. Then, the value common difference of the $(i+1)$st block is the next value in the arithmetic progression of the ith block. And the difference between the largest value of the ith block and the smallest of the $(i+1)$st block is the common difference of the $(i+1)$st block.

That is, let $B_i = \{u_i + tv_i : 0 \leq t < q_i\}$ for $i \in [s]$, where $u_1 = v_1 = 1$ and $v_{i+1} = u_i + q_i v_i$ and $u_{i+1} = u_i + (q_i - 1)v_i + v_{i+1}$. Define the stamp basis associated to $q_1, \ldots, q_s$ as $G(q_1, \ldots, q_s) = B_1 \cup \cdots \cup B_s$.

Alter and Barnett's constructions is the case $q_1 = \cdots = q_{s-1} = q$ and $q_s = q + r$ where $k = qs + r$ is the number of denomination. Our improvement is the case $q_1 = \cdots = q_{s-r} = q$ and $q_{s-r+1} = \cdots = q_s = q + 1$.

▶ **Proposition 18.** *Let $q_1, \ldots, q_s \geq 2$ 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 $n_{i+1}(G_i) \geq u_{i+1} - 1$. 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

[5] https://github.com/openfheorg/openfhe-development
[6] The timings are obtained as a mean of 100 products.

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 $m \in \{u_i, \dots, v_{i+1} - 1\}$. 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, $m - (u_i + tv_i)$ 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 $m \in \{v_{i+1}, \dots, u_{i+1} - 1\}$. 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. $u_i = u_{i-1} + (q_{i-1} - 1)v_{i-1} + v_i$.

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 does 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 $(a_i, b_i), (a_{i+1}, b_{i+1})$ 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 $(a_{i-1} + 1, b_{i-1} + (q_i - 1)), (a_i - 1, b_i + 1)$ (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 $q_j \leq b_j \leq (q_j - 1)a_j$). 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 $n_s(G_k) \leq v_{s+1} - 1$. ◀

▶ **Proposition 5** (). *The k-range of F_k is $n_k(F_k) = f_{2k+1} - 1$.*

Proof of Proposition 5. We first prove by induction on k that $n_k(F_k) \geq f_{2k+1} - 1$. For $k = 1$, $n_1(F_1) = n_1(\{1\}) = 1 = f_3 - 1$.

Consider now $k > 1$. By induction hypothesis, $n_k(F_{k-1}) \geq f_{2k-1} - 1$. We prove that every integer $m < f_{2k+1}$ can be written as $\sum_{i=1}^{k} \lambda_i f_{2i}$ with $\sum_i \lambda_i \leq k$. If $m < f_{2k-1}$, this is true by induction hypothesis. If $f_{2k-1} \leq m < f_{2k}$, $m - f_{2k-2} < f_{2k} - f_{2k-2} = f_{2k-1}$. Therefore, $m - f_{2k-2}$ can be written with $(k-1)$ stamps by induction hypothesis. Hence m can be written with k stamps with the additional stamp f_{2k-2}. Similarly if $f_{2k} \leq m < f_{2k+1}$, $m - f_{2k} < f_{2k+1} - f_{2k} = f_{2k-1}$ and m can be obtained using the stamp f_{2k} and $(k-1)$ other stamps.

We now prove that $n_k(F_k) < f_{2k+1}$, that is f_{2k+1} cannot be written as $\sum_{i=1}^{k} \lambda_i f_{2i}$ with $\sum_i \lambda_i \leq k$. We assume otherwise and rewrite the sum as $f_{2k+1} = \sum_{i=1}^{2k} \lambda_i f_i$ where odd indexed λ_is vanish. We describe a rewriting rule that keeps $\sum_i \lambda_i$ non-increasing, and let the following property invariant:

($\mathcal{P}$) let $j = \max\{i : \lambda_i > 1\}$, then $\lambda_{j+1} = 0$, $\lambda_i \in \{0, 1\}$ and $\lambda_i \lambda_{i+1} = 0$ for $i > j + 1$, and every odd indexed λ_i for $i < j$ vanishes. (If no such j exists, $j = 0$ by convention.)

The rewriting rule is as follows: Consider $\lambda_j > 1$ as in ($\mathcal{P}$). Using the identity $2f_j = f_{j-2} + f_{j+1}$, we replace λ_j by $\lambda_j - 2$, increase λ_{j-2} and set $\lambda_{j+1} = 1$. If $\lambda_{j+2} = 1$, we use the identity $f_{j+1} + f_{j+2} = f_{j+3}$ to set $\lambda_{j+1} = \lambda_{j+2} = 0$ and $\lambda_{j+3} = 1$, and so on until no two consecutive λ_is, $i > j + 1$, are non-zero. If $\lambda_{j+2} = 0$ but λ_j is still non-zero, we apply the same process from the pair $(\lambda_j, \lambda_{j+1})$. It is readily checked that property ($\mathcal{P}$) holds after one step of rewriting.

We apply the rewriting rule until one of the two events occur: Either no j such that $\lambda_j > 1$ exists, or $j = 2$. In the first case, f_{2k+1} is a sum of non-consecutive non-repeated Fibonacci numbers. This is impossible by Zeckendorf theorem, or simply by noting that the largest such sum is $\sum_{i=1}^{k} f_{2i} = f_{2k+1} - 1$. In the second case, $f_{2k+1} = \lambda_2 f_2 + \sum_{i \geq 4} \lambda_i f_i$ with $\sum_i \lambda_i \leq k$

and $\lambda_2 > 1$. This implies that $\lambda_i = 0$ for $i > 2k$. Since there are at most $k - \lambda_2$ non-zero terms in the sum $\sum_{i\geq 4} \lambda_i f_i$, its value is at most $\sum_{\ell=\lambda_2+1}^{k} f_{2\ell} = f_{2k+1} - f_{2(\lambda_2+1)+1}$. But $\lambda_2 < f_{2(\lambda_2+1)+1)}$ and the sum cannot reach f_{2k+1}. In both cases, we obtain a contradiction. ◀

▶ **Corollary 6** (). *For $k \leq s$, let $s = qk + r$ by Euclidean division, then: $\left\lceil \frac{s}{\sqrt{5}} \right\rceil (1+\varphi)^k > n_s(F_k) > \left(\lfloor \frac{s}{k} \rfloor \left(1 - \frac{\sqrt{5}}{\varphi^{2k+1}} \right) + \frac{1}{\varphi^{2k-2r}} \right) \frac{\varphi}{\sqrt{5}} (1+\varphi)^k$.*

Proof of Corollary 6. For the upper bound, by Lemma 4 with basis F_k, we get $n_s(F_k) < s f_{2k}$. Then, by Equation (1), $f_{2k} = \left\lceil \frac{1}{\sqrt{5}} \varphi^{2k} \right\rceil = \left\lceil \frac{1}{\sqrt{5}} (1+\varphi)^k \right\rceil$. For the lower bound, consider that each group of k stamps can reach $f_{2k+1} - 1$ by Corollary 6. Then the last group with r stamps can reach at least $f_{2r+1} - 1$. Therefore, at least all integers up to $q(f_{2k+1} - 1) + f_{2r+1} - 1$ can be reached. This is $\left(q - \frac{1}{f_{2k+1}} + \frac{f_{2r+1}}{f_{2k+1}} \right) f_{2k+1} - 1$, from which one gets the announced lower bound using Equation (1), with $q = \lfloor \frac{s}{k} \rfloor$. ◀

▶ **Proposition 7** (). *Using the variant of Alter and Barnett's construction with r blocks of size $(q+1)$ and then $(s-r)$ blocks of size q, the dominant term of $n_s(G_k)$ becomes $(1 + (q+1)(1+\epsilon_{q+1}))^r (1 + q(1+\epsilon_q))^{s-r}$.*

Proof of Proposition 7. Use Equation (2) with $q \leftarrow q+1$ and $s \leftarrow r$, for the first r blocks; then use Equation (2) again, now with q and $(U_0, U_1) = (U_{r-1}, U_r)$ from the previous blocks. ◀

▶ **Proposition 8** (). $n_{s_1+s_2}(k_1+k_2) \geq (n_{s_1}(k_1)+1)(n_{s_2}(k_2)+1) - 1$

Proof of Proposition 8. We recall the proof of [19]. 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. $C_k = A_{k_1} \| (n_1+1)B_{k_2}$. It is easy to see that we can generate $i(n_1+1)$ for $i \in [n_2]$ 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 $n_s(k) \geq n_s(C_k)$. ◀

▶ **Lemma 9** (). *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 of Lemma 9. 1. First, we have that $T = 1 + T - 1 = (1+T)^{2^0} - 1$;
2. Then, by induction, we directly obtain that $((1+T)^{2^i} - 1 + 1)^2 - 1 = (1+T)^{2^{i+1}} - 1$. ◀

▶ **Lemma 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 of Lemma 11. 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 $t = k/v$. With this, we get 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. ◀

In Figure 3, we compare the recursive algorithm and the Fibonacci sequence, using the programs `./bin/basis` and `./bin/fibo` of the GStamps library.

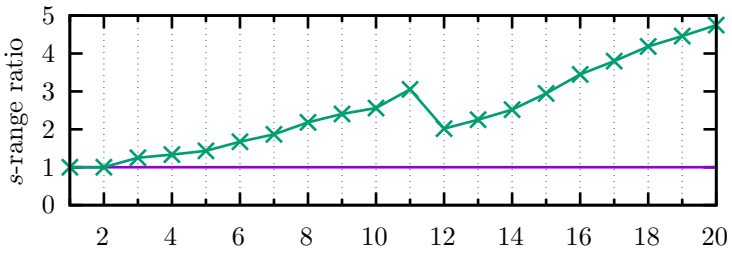

Figure 3 Range ratio between the recursive algorithm and the Fibonacci sequence, for $k = s$

A.3 Proofs and figures of Section 3

For the sake of completeness we give in Algorithm 2, a version of the algorithm of [16].

Algorithm 2 Mossige s-range [16]

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

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

▶ **Proposition 15** (). *There exists an LPSP algorithm which requires $O(ksa_k)$ assignments or more precisely $O(kn_s(A_k) + sa_k)$.*

Proof of Proposition 15. We exhibit the following Algorithm 3 and prove its correctness and complexity bounds.

The correctness of Algorithm 3 follows the intuition given in the introduction of Section 3.2. More precisely, before showing the optimality of $T[i]$, we note that it is easy to show by a simple induction 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

Algorithm 3 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
  i ← i + 1
until T[i] > s
return i − 1
```

$T[i] < s$, 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.

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 term 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. It is easy to see that 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. And it is also easy to see that $T[t]$ 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, which is absurd since $x > a_l$ and we 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.

The complexity bound is trivial to prove: 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 $O(ksa_k)$. ◀

▶ **Proposition 16 ().** *Algorithm 1 is correct and requires $O(ksa_k)$ assignments or more precisely $O(kn_s(k) + a_k)$.*

Proof of Proposition 16. The correction of this algorithm is a direct consequence of the correction of Proposition 15, combined with the fact that in Proposition 15 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 15, 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. ◀

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

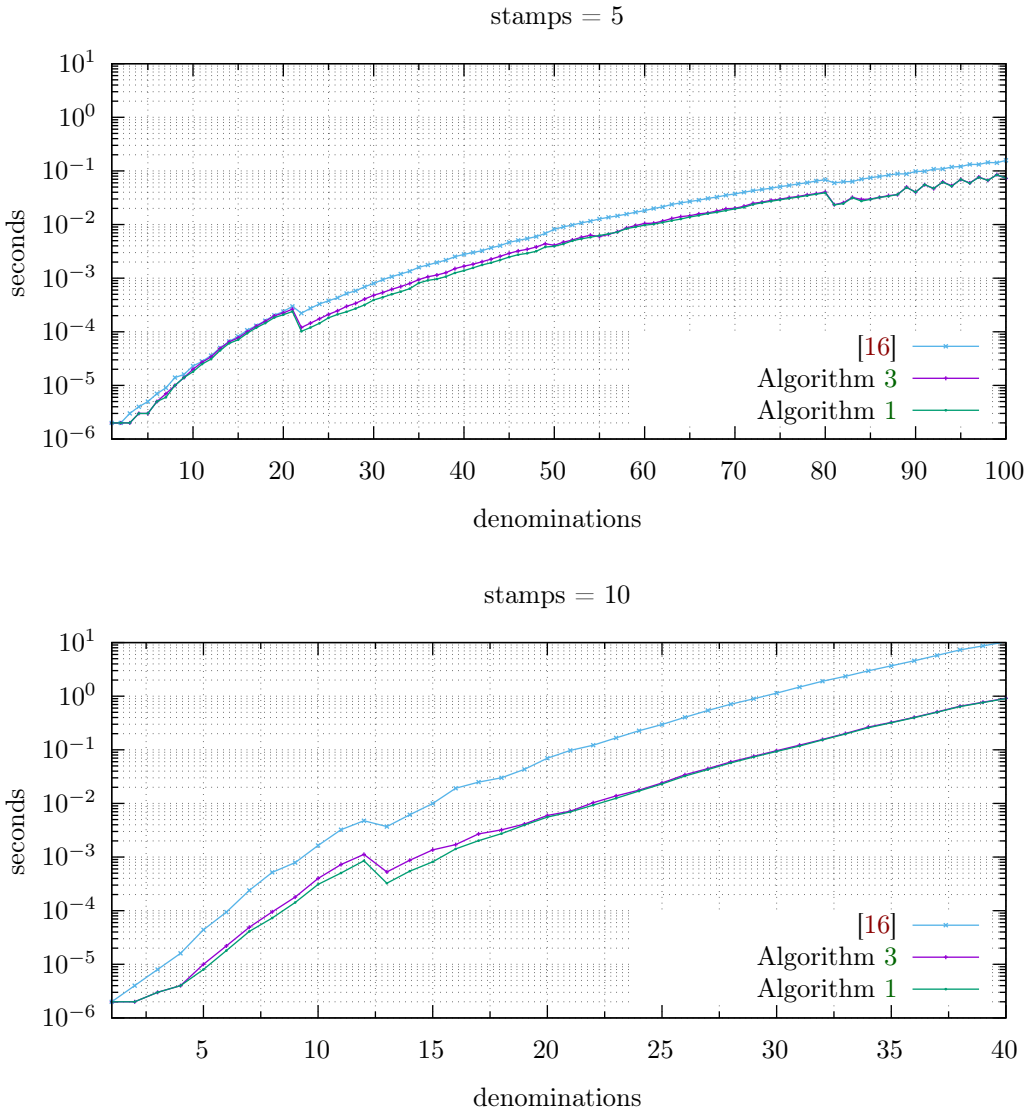

Figure 4 LPSP algorithms, $s = 5$ and $s = 10$

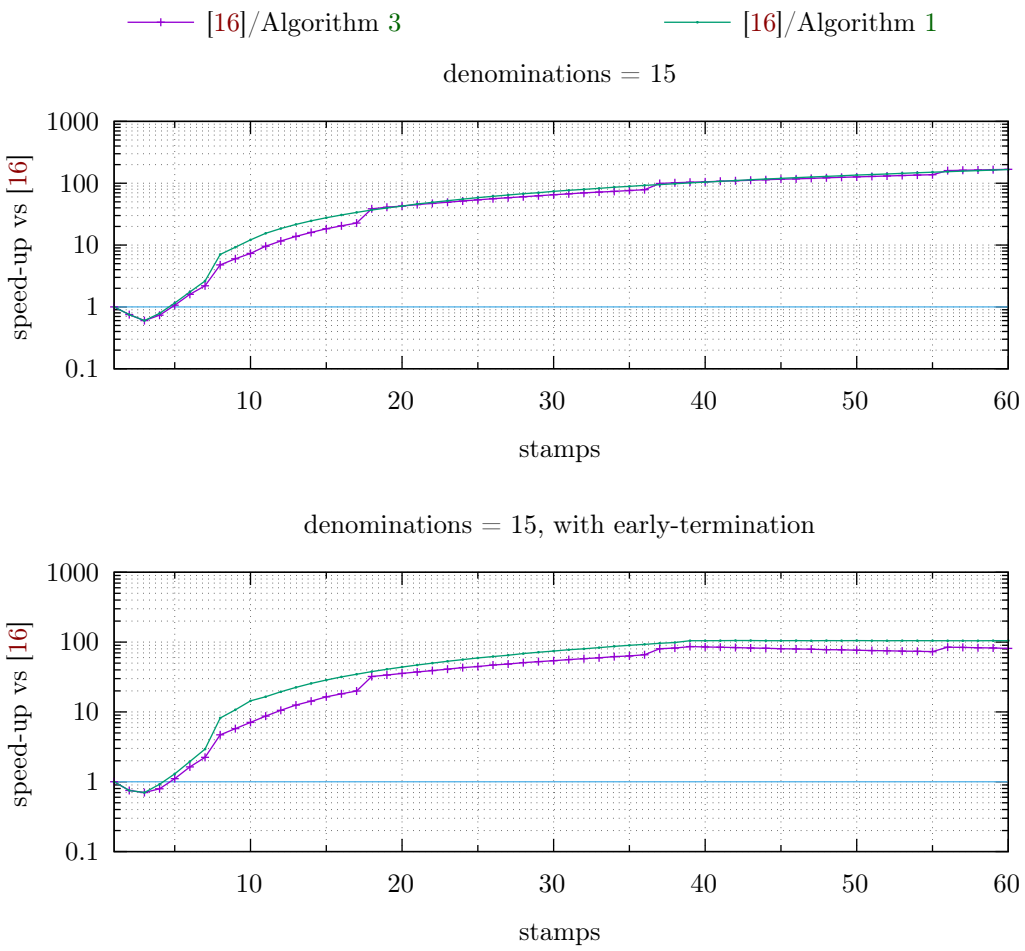

Figure 5 LPSP algorithms, $k = 15$, early terminated by Selmer's lemma [22] (effective for $s \geq 40$).

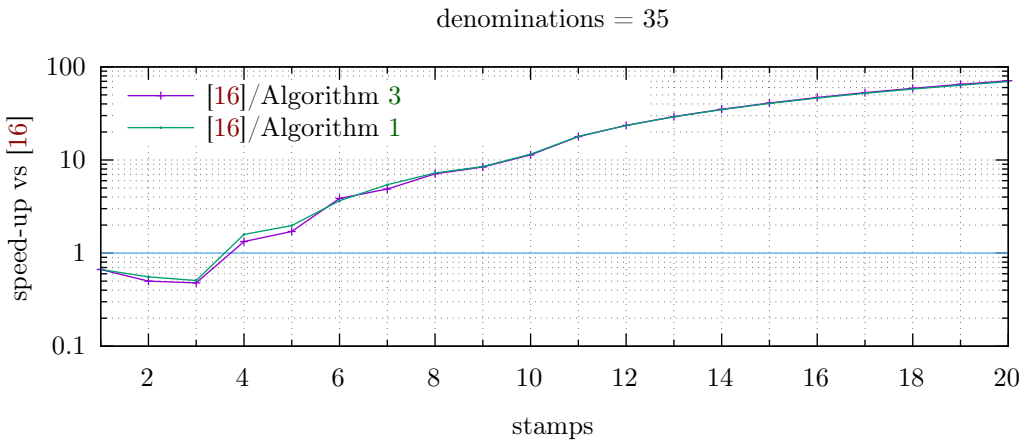

Figure 6 LPSP algorithms, $k = 35$