Sieving for large twin smooth integers using single solutions to Prouhet-Tarry-Escott

Knud Ahrens

Faculty of Computer Science and Mathematics
University of Passau, Germany
knud.ahrens@uni-passau.de

Abstract. In the isogeny-based track of post-quantum cryptography the signature scheme SQISign relies on primes p such that $p\pm1$ is smooth. In 2021 a new approach to find those numbers was discovered using solutions to the Prouhet-Tarry-Escott (PTE) problem. With these solutions one can sieve for smooth integers A and B with a difference of $|A-B| = C$ fixed by the solution. Then some $2A/C$ and $2B/C$ are smooth integers hopefully enclosing a prime. They took many different PTE solutions and combined them into a tree to process them more efficiently. But for bigger numbers there are fewer promising PTE solutions so their advantage over the naive approach (checking a single solution at a time) fades.
For a single PTE solution the search can be optimised for the corresponding C and allows to only sieve those integers that are divisible by C. In this work we investigate such optimisations and show a significant speed-up compared to the naive approach.

Keywords: Isogeny-based cryptography · post-quantum cryptography · twin smooth integers · Prouhet-Tarry-Escott problem · SQISign.

1 Introduction

Cryptography based on isogenies of elliptic curves produced some candidates for post-quantum cryptography. One example from this field is the signature scheme SQISign [8]. Isogenies of small degree can be evaluated efficiently using Vélu's formulae, while isogenies of composite degree can be decomposed into several isogenies of prime degree. Therefore many protocols based on supersingular curves use fields of characteristic p such that either $p + 1$ or $p - 1$ is smooth, which allows for efficient evaluation of isogenies of degree $p + 1$ or $p - 1$, respectively. A smooth integer only has small prime factors below a fixed smoothness bound.

SQISign uses isogenies of degree $p + 1$ and $p - 1$, and to be able to evaluate them efficiently one needs both $p + 1$ and $p - 1$ to be smooth in which case p is called a twin smooth prime. There are approaches to find such primes using the extended Euclidean algorithm or polynomials of the form $x^n - 1$, but in 2021 Costello, Meyer and Naehrig [6] found a new way using solutions to the Prouhet-Tarry-Escott (PTE) problem and produced integers with lower

smoothness bounds. They used several PTE solutions at once and optimised for a setting with many solutions. We will only use a single PTE solution at a time and customise the sieve accordingly. This is advantageous when only a few suitable solutions are available.

The rest of this paper is structured as follows. First we summarise the main idea of Costello, Meyer and Naehrig [6]. Then we motivate our optimisation for a single solution and show how it can be done theoretically. Next we discuss some caveats for implementation and compare our experimental results to the naive and the multi-solution tree approach. A Sage as well as a C implementation are available at https://git.fim.uni-passau.de/ahrens/twin-smooth-integers. The experimental results are then backed up with an approximation and a bound. Finally there is a short conclusion including some questions for future work.

2 Sieving with a PTE solution

In this section we shortly present the Prouhet-Tarry-Escott (PTE) problem and how [6] used it to find twin smooth integers.

2.1 The Prouhet-Tarry-Escott problem

The Prouhet-Tarry-Escott problem of degree[1] n and integer k is a set of equations

$$a_1^j + \cdots + a_n^j = b_1^j + \cdots + b_n^j \quad \text{with} \quad \{a_1, \ldots, a_n\} \cap \{b_1, \ldots, b_n\} = \emptyset$$

for $1 \leq j \leq k$ and $a_i, b_i \in \mathbb{Z}$. A solution $\mathcal{A} = \{a_1, \ldots, a_n\}$, $\mathcal{B} = \{b_1, \ldots, b_n\}$ with $k = n - 1$ is called an ideal solution. There are several known solutions and even some parametric solutions, but for $n = 11$ and $n \geq 13$ no ideal solutions are known [4].

A statement from Borwein and Ingalls [2] using Newton's identities tells us that for each ideal solution the difference $C = |a(x) - b(x)|$ of the corresponding polynomials

$$a(x) = \prod_{i=1}^{n}(x - a_i), \quad b(x) = \prod_{i=1}^{n}(x - b_i) \tag{1}$$

is an integer constant. Therefore $a(x)/C$ and $b(x)/C$ give us consecutive integers every time $a(x)$ (or $b(x)$ equivalently) evaluates to a multiple of C and we hope that $2a(x)/C$ and $2b(x)/C$ are smooth with a prime in between. Since the polynomials already have n factors, chances are high that the evaluations are smooth. Ideal solutions with small differences C seem to produce a higher rate of twin smooth integers, but their number is limited (see [6, table 2]) since the C in parametric solutions often increases fast.

[1] Since we will mostly deal with ideal solutions and the polynomials in equation (1), we deviate from the usual notation of size n and degree k.

The paper [6, table 3] shows heuristically that this approach has a higher chance of success for a given smoothness bound or rather allows for a lower smoothness bound for a given probability to find twin smooth integers compared to previous attempts, e.g. using the extended Euclidean algorithm.[2]

2.2 The sieve

As we have just seen, we can use solutions of the PTE problem to find twin smooth integers. So we need an ideal PTE solution and then for different $\ell \in \mathbb{Z}$ check if $a(\ell)$ and $b(\ell)$ are smooth and if $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$. For ease of notation an ideal PTE solution will only be called a solution in the following.

Some solutions can be found in a database by Shuwen [10] and there are constructions to find new ones. Costello, Meyer and Naehrig [6, section 3.2] mention some constructions and give several solutions in their paper and on GitHub.

To check smoothness we basically use a sieve of Eratosthenes on the interval $I \subset \mathbb{Z}$ with different powers of p instead of different primes. We start by initialising an array with $(i, 1)$ for i in I. For all primes p below the smoothness bound we check if i is divisible by p and if so multiply the second entry of the tuple with p. Then for all multiples of p^2 the second entry is multiplied by p and so on until p^e is bigger than the numbers in the interval. Finally when this is done for all powers of all p, then the elements where both entries are identical are the smooth numbers.

The smoothness of $a(\ell)$ and $b(\ell)$ is then tested using their factors. Let $\{a_1, \ldots, a_n, b_1, \ldots, b_n\}$ be the set of roots of $a(x)$ and $b(x)$. Note that these roots are small compared to ℓ. Then for each integer ℓ in the search interval I we look up the values $\{\ell - a_1, \ldots, \ell - a_n, \ell - b_1, \ldots, \ell - b_n\}$. If all of them are smooth, then surely their product is smooth and we have two smooth integers $a(\ell)$ and $b(\ell)$ that differ by C. For those smooth integers we finally compute $a(\ell) \bmod C$ (or $b(\ell) \bmod C$) to see if $a(\ell)/C$ and $b(\ell)/C$ are integers. If they are, we have found twin smooth integers and can check if $a(\ell)/C + b(\ell)/C$ is a prime.

If there are multiple PTE solutions of the same degree n one may speed up the search as follows. All PTE solutions can be normalised in a way such that 0 is always in the zero set of the corresponding polynomials and from now on all solutions are taken to be normalised. So if $\ell - 0$ is not smooth, then no solution will give smooth integers $a(\ell), b(\ell)$. Similarly if there are different (normalised) solutions their corresponding polynomials probably share some zeros. If $\ell - r$ is not smooth, we can skip all solutions that share this zero r for this ℓ. Based on this Costello, Meyer and Naehrig [6] presented a tree structure based on a hitting set problem with the most common zeros as vertices to test many solutions at the same time with the minimal number of look-ups in the list produced by the sieve. When smooth integers $a(\ell), b(\ell)$ are found, the residues modulo their corresponding C are calculated to see if they give twin smooth integers.

[2] There has been a recent publication [3] by the same authors with a different approach.

For $n = 6$ there are 2438 solutions with $C \leq 2^{50}$, so if all of them can be checked by only a few look-ups this is a significant speed-up compared to the naive approach testing every solution separately. More on their algorithm and a detailed example can be found in their paper [6] alongside a link to their code on GitHub.

3 Optimising for single PTE solutions

Since different solutions have different C, the tree approach forbids to do the modulo calculation first, but for a single solution this is possible. Costello, Meyer and Naehrig [6] left it as an open question to explore this path. In this section we will see why this is relevant and how it can be done efficiently. We will also discuss implementations and some experimental results.

3.1 Motivation and parameters

For primes in the range of 256 bit $n = 6$ is the sweet spot, but solutions of higher degree n allow for much lower smoothness bounds for larger primes at a given probability to find twin smooth integers [6, table 3]. Or put the other way: when we need a twin smooth integer in the 512 bit range with smoothness bound below 2^{25} we probably need to test 2^{50} values before we get a hit when using $n = 6$ compared to only 2^{30} for $n = 8$. According to [6, table 3] a search for twin smooth primes in the range of 512 bits and beyond therefore favours solutions of higher degree and [6, table 2] shows that only a few solutions with high degree n and small C are known. Therefore checking them individually is feasible.

The idea is to calculate the set R of all elements r in $[0, C)$ that solve $a(r) \equiv b(r) \equiv 0 \bmod C$. Obviously $a(r + C) \equiv a(r) \bmod C$ and thus all relevant $\ell \in I$, i.e. those solving $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$, are of the form $\ell = r + kC$. Hence, instead of looking at all ℓ in the search interval I it is sufficient to look at $\{r + kC \mid r \in R,\, k \in \mathbb{Z}\} \cap I$. This subset has size about $(|R|/C) \cdot |I|$ and can be significantly smaller than I. Table 2 and the heuristic in section 4.3 indicate that this fraction decreases for larger C. Another improvement is that the modulo computation only needs to be done once per PTE solution and not for every smooth integer $a(\ell)$ (or $b(\ell)$) anew.

To see more clearly why we want a larger degree n and a small difference C (which is specific to every PTE solution) we have to understand how they influence our search. On the one hand a higher number n of factors improves the chances of $a(\ell)$ and $b(\ell)$ being smooth, on the other hand it decreases the number of possible values for ℓ. We can approximate p with

$$p = \frac{a(\ell) + b(\ell)}{C} \approx \frac{2a(\ell)}{C} \approx \frac{2b(\ell)}{C} \approx 2\frac{\ell^n}{C}.$$

For the number of bits $\log_2 p$ in the range of $x \pm \delta$ we get $\log_2 \ell$ roughly in the range $(x \pm \delta - 1 + \log_2 C)/n$. We see that C does not really change the size of the

search interval, it is merely a shift, but higher values for n decrease the search space. This is why some entries in [6, table 3] are greyed out, but this is more relevant when searching for smaller primes.

As we mentioned above, solutions with larger C for a fixed n in general have a smaller fraction $|R|/C$ of $\ell \in I$ that satisfy $a(\ell) \equiv b(\ell) \equiv 0 \bmod C$. This makes our approach faster compared to the naive one, but not more efficient in absolute terms. A smaller density $|R|/C$ makes it less likely to find a twin smooth integer, or a twin smooth prime accordingly, in a given interval I. Increasing I could help but it has two drawbacks. Firstly calculating smoothness (for more elements) is costly and secondly the interval could be restricted such that enlargement is not possible. Thus PTE solutions with $n > 6$ and small C have a better chance of finding twin smooth primes of size at least 512 bit.

3.2 Computing congruences

As we have seen, we need to find the set of relevant residues $R = \{r \in \mathbb{Z}/C\mathbb{Z} \mid a(r) \equiv b(r) \equiv 0\}$. Calculating $a(r) \bmod C$ (or $b(r)$) for all $0 \leq r < C$ becomes too slow quite soon. Using the Chinese Remainder Theorem can improve this, but it also has a downside: the resulting residue classes modulo C have no predictable order. If we write them in a list we can order it afterwards, but the list can still get long and this pre-computation might take some time.

As long as C is smaller than $|I|$ one could calculate them separately on the fly and check one set $\{r + kC \mid k \in \mathbb{Z}\} \cap I$ at a time. But if we can efficiently use a look-up table of size $|I|$ for the smoothness, then we can also efficiently use a table of size $|R| \leq C \leq |I|$ for the relevant residue classes. Moreover, we can pre-compute R and then use it for several intervals, whereas it has to be computed every time when we do it on the fly and we still need the list of residue classes modulo all prime power factors of C.

As soon as C and especially when $|R|$ becomes greater than $|I|$, the advantage over the naive approach disappears when we still consider all $r \in R$. This would be the case if we calculated the CRT on the fly. If we have an ordered list $r_1 < \cdots < r_{|R|}$ of residues, we can find the smallest representative in the interval (for example using nested intervals) and then take the next one until we leave the interval. This way we still only need to check about $|R|/C$ of the whole interval. Here we need to keep in mind that we could wrap around from $r_{|R|} + kC$ to $r_1 + (k+1)C$. To handle this in the implementation we can calculate a number of relevant steps s such that the difference $r_{i+s} - r_i$ (or $r_{i+s-|R|} + C - r_i$ in the case of a wrap-around) is greater than the size of I for all initial elements $r_i \in R$. Then we know in advance if we have to wrap around.

The evaluations $a(r), b(r)$ can become large. Since we know the factorisation it is useful to compute the individual factors $r - a_i$ (or $r - b_i$) and multiply them reducing modulo C after every step. If we allow C to have 60 bits, then the intermediate steps can have more than 64 bits. Therefore we need to introduce 128 bit variables in the implementation.

3.3 Sieving for smoothness is still optimal

When using only one solution, we have seen that it suffices to check smoothness for $a(\ell), b(\ell)$ with $\ell \in \{r + kC \mid r \in R,\, k \in \mathbb{Z}\} \cap I$. Let Z be the set of roots of the polynomials a, b. Since we verify smoothness of $a(\ell)$ by looking at the factors $\ell - a_i$, we have to calculate the smoothness of $\{r + kC - z_i \mid r \in R,\, z_i \in Z,\, k \in \mathbb{Z}\} \cap I$. Now there are several options to consider.

There is an algorithm by Bernstein [1] to test smoothness for an arbitrary set. It computes the product P of all primes below the smoothness bound and then basically calculates repeated squares of this product modulo ℓ up to $P^{2^e} \bmod \ell$ for each element ℓ of the set where e is the smallest integer such that $2^{2^e} \geq \ell$. This is only faster than the sieve for small sets and sets of small numbers. For elements of size 2^{64} one needs to compute up to 6 squares modulo ℓ per element in the set. So the running time of this test depends on the size of the elements and the size of the set.

The sieve has a lower bound on the running time that depends solely on the smoothness bound, because it runs through all relevant powers of all primes below the bound, so there is a fixed number of loops that have to be executed at least once. But above this threshold it is linear in the size of the interval and does not depend on the size of the elements. Since the elements we are interested in are usually significantly larger than the smoothness bound, we are above this threshold. The only operations are additions and some multiplications, which are easy and fast to compute.

The logarithm allows us to replace multiplications with additions and taking only integer approximations makes the logarithm efficient. The loop for p^j has I/p^j iterations. Therefore the smallest p^j have the biggest computational impact, while contributing only p or $\log p$ to the product or sum, respectively. For example we have 2^{19} multiplications for 2 in an interval of size 2^{20}. If we can tolerate a non-smooth factor, then we can safely omit a factor of 2 and write it off as non-smooth part. Since the SQISign protocol tolerates such a non-smooth factor, the modification to use rounded logarithms and to ignore the smallest primes or at least their small powers is a good option to speed up this part of the calculations.

We could also use a polynomial sieve to check the set $f(I)$ instead of an interval I for any polynomial $f \in \mathbb{Z}[X]$. Let $r \in R$, k be an integer and $z_i \in Z$ as above, then we can try to test only the relevant elements $r + kC - z_i \in I$ using the polynomials in $\mathbb{Z}[X]$ defined by $f_{r,z_i}(x) = Cx + r - z_i$. The problem here is, that the polynomial sieve requires us to solve $f_{r,z_i}(x) = 0 \bmod q$ for every prime power q, i.e. in every loop. And on top we would need $2n|R|$ different sieves (one for every polynomial) or use $a(Cx + r)$ and $b(Cx + r)$ instead of f_{r,z_i}. But then we would still have $2|R|$ sieves and had to solve a polynomial equation of degree n in a ring with zero divisors in every loop. This is clearly slower than sieving everything with a regular sieve. More information on sieving can be found in [7].

So we will keep using the standard or the logarithmic sieve to check every element in the interval for smoothness before applying either the tree, the naive or our new approach.

3.4 Experimental results

The running times of Sage and C implementations for different PTE solutions are given in the appendix. The target size of the integers $a(\ell)/C$ and $b(\ell)/C$ is 2^{240} to 2^{256} for a 256 bit prime, 2^{370} to 2^{384} for a 384 bit prime and 2^{500} to 2^{512} for a 512 bit prime. The smoothness bounds are taken from [6, table 3] for a probability of 2^{-30}. The start values depend on the degree n of the PTE solution, the size of C and the target size (cf. section 3.1). The search interval I is separated into chunks of size 2^{20} for the sieving. The solutions used for $n = 6$ all have double roots and the last two only coincidently have the same C value.

For an interval of fixed size the naive algorithm has similar running times for different solutions and seems to depend more on the number of smooth integers in the interval. This number is higher for larger smoothness bounds and smaller start values. This is due to the fact that for every smooth ℓ the other factors $\ell - a_i$ and $\ell - b_i$ of $a(\ell)$ and $b(\ell)$, respectively, have to be checked until one of them is not smooth or two smooth integers are found.

The optimised algorithm depends heavily on the ratio of relevant residues $|R|/C$ and there also seems to be a dependency on the number of smooth numbers in the interval. It is always significantly faster than the naive approach and probably even faster than the tree approach for higher degrees n.

Due to memory restrictions for larger R not all PTE solutions for $n \geq 7$ have been tested. For $n = 7$ we need less than 1/20 of the time of the naive approach and only 10 solutions[3] with $C \leq 60$ are known [6, table 2]. So this approach is definitively faster. For $n \geq 9$ there are only one or two solutions with small C, so the tree is at most twice as fast as the naive algorithm. Judging from the heuristic in section 4 and the other results, the optimised approach will be more than twice as fast as the naive approach and therefore faster than the tree approach.

4 Roots mod p^e

In this section we will give a formula to compute the (maximal) number of roots of a given polynomial modulo prime powers. Let p be a prime and $f(x) = \prod_{i=1}^{n}(x - z_i)$ a polynomial with integer roots z_i and degree n.

Since all solutions of $f(x) \equiv 0 \bmod p^e$ also have to solve this equation modulo p, we only need to look at integers that are congruent to one of the roots z_i. Let $S = \{\sum_{j \in \mathbb{N}_{>0}} d_j p^j \mid d_j \in \mathbb{Z},\ 0 \leq d_j < p\}$, then $z + S$ is the set of all integers that are congruent to z modulo p. The p-adic valuation ν_p gives the highest power of p that divides the argument and $\nu_p(0)$ is infinite by definition.

4.1 For one root

First we will only look at one fixed root z of f. Let $m_e(z)$ be the number of roots z_i of f that are congruent to z modulo p^e. Then $1 \leq m_e(z) \leq n$ and note that

[3] We found two solutions that were not included in [6]. One was found in [10] and the other is a parametric solution from [5] for $m = -3$. They are listed online.

$m_0(z) = n$ for all roots of f. Recall that $\nu_p(z-z) = \infty$ and let $m_\infty(z)$ be the number of roots that are equal to z. Since we fixed z for this subsection we will write just m_e and use $v_i = \nu_p(z - z_i)$ as shorthand. Without loss of generality let $z = z_1$ and $v_i \geq v_j$ for $1 \leq i \leq j \leq n$. For $i > m_\infty$ we define δ_i via $p^{v_i}\delta_i = z - z_i$, then $\nu_p(\delta_i) = 0$.

When we look at the possible solutions $x = z + s$ to $f(x) \equiv 0 \bmod p^e$ above z with $s \in S$ we get

$$\begin{aligned} f(z+s) &= \prod_{i=1}^{n}\big(z - z_i + s\big) = \prod_{i=1}^{m_\infty}\big(0+s\big)\prod_{i=m_\infty+1}^{m_0}\big(p^{v_i}\delta_i + s\big) \\ &= p^{m_1}\prod_{i=1}^{m_\infty}\big(0 + \sum_{j\geq 1} d_j p^{j-1}\big)\prod_{i=m_\infty+1}^{m_2}\big(p^{v_i-1}\delta_i + \sum_{j\geq 1} d_j p^{j-1}\big) \\ &\quad\cdot \prod_{i=m_2+1}^{m_1}\big(\delta_i + \sum_{j\geq 1} d_j p^{j-1}\big)\prod_{i=m_1+1}^{m_0}\big(\delta_i + s\big) \\ &\equiv p^{m_1}\prod_{i=1}^{m_2}\big(d_1\big)\prod_{i=m_2+1}^{m_1}\big(\delta_i + d_1\big)\prod_{i=m_1+1}^{m_0}\big(\delta_i\big) \mod p^{m_1+1}. \end{aligned}$$

So for $f(z+s)$ to be zero modulo p^{m_1+1}, we need d_1 to be either 0 or $d_1 \equiv -\delta_i \bmod p$ for $m_2 < i \leq m_1$. Then $z + d_1 p \equiv z - z + z_i = z_i \bmod p^2$ for $d_1 \neq 0$ and we will treat elements $z + s$ with $d_1 \equiv -\delta_i \bmod p$ as belonging to the corresponding z_i. That allows us to set $d_1 = 0$ for all solutions $s \in S$ to $f(z+s) \equiv 0 \bmod p^e$ with $e > m_1$.

For an inductive argument assume we can set $d_1, \ldots, d_{k-1}$ to zero for all solutions above z modulo p^e with $e > \sum_{i=1}^{k-1} m_i$. Then $s \in S$ is of the form $s = \sum_{j\geq k} d_j p^j$ and

$$\begin{aligned} f(z+s) &= \prod_{i=1}^{n}\big(z - z_i + s\big) = \prod_{i=1}^{m_\infty}\big(0+s\big)\prod_{i=m_\infty+1}^{m_0}\big(p^{v_i}\delta_i + s\big) \\ &= p^{km_k}\prod_{i=1}^{m_\infty}\big(0 + \sum_{j\geq k} d_j p^{j-k}\big)\prod_{i=m_\infty+1}^{m_k}\big(p^{v_i-k}\delta_i + \sum_{j\geq k} d_j p^{j-k}\big) \\ &\quad\cdot \prod_{\ell=1}^{k} p^{(k-\ell)(m_{k-\ell}-m_{k-\ell+1})}\prod_{i=m_{k-\ell+1}+1}^{m_{k-\ell}}\big(\delta_i + \sum_{j\geq k} d_j p^{j-k+\ell}\big) \\ &\equiv p^M \prod_{i=1}^{m_{k+1}}\big(d_k\big)\prod_{i=m_{k+1}+1}^{m_k}\big(\delta_i + d_k\big)\prod_{i=m_k+1}^{m_0}\big(\delta_i\big) \mod p^{M+1} \end{aligned}$$

where $M = km_k + \sum_{\ell=1}^{k}(k-\ell)(m_{k-\ell} - m_{k-\ell+1}) = \sum_{\ell=1}^{k} m_\ell$. The above only vanishes modulo p^{M+1} for $d_k = 0$ or $d_k \equiv -\delta_i \bmod p$ with $m_{k+1} < i \leq m_k$. Therefore $z + d_k p^k \equiv z - z + z_i = z_i \bmod p^{k+1}$ for $d_k \neq 0$ and we will again treat elements $z + s$ with $d_k \equiv -\delta_i \bmod p$ as belonging to the corresponding z_i. As before this allows us to set d_k to zero for all solutions modulo p^e with $e > M = \sum_{\ell=1}^{k} m_\ell$. So by induction we can see that this is true for all $k \in \mathbb{N}$.

4.2 Number of solutions

In this subsection we will use the results of the previous subsection to calculate the number of solutions $x \in \mathbb{Z}/p^e\mathbb{Z}$ of $f(x) = 0$.

For all $z+s \in \mathbb{Z}/p^e\mathbb{Z}$ we have s of the form $s = \sum_{i=1}^{e-1} d_i p^i$ and since $0 \leq d_i < p$ we can have at most p^{e-1} solutions above z modulo p^e. For $1 \leq e \leq m_1(z)$ we found that $f(z+s)$ is always zero modulo p^e. Therefore all d_i are arbitrary and there are p^{e-1} solutions above z. Starting from $e = m_1(z)+1$ we set d_1 to zero and by induction we can set d_i to zero for all $e > \sum_{\ell=1}^{i} m_\ell(z)$. This shows that there are p^{e-j} solutions above z modulo p^e for $\sum_{\ell=1}^{j-1} m_\ell(z) < e \leq \sum_{\ell=1}^{j} m_\ell(z)$.

If z is a single root then there exists an $e_{\text{fin}} = \max\{e \in \mathbb{N} \mid m_e(z) > 1\}$. Let $M = \sum_{\ell=1}^{e_{\text{fin}}} m_\ell(z)$ then there are $p^{M-e_{\text{fin}}}$ solutions above z modulo p^M. Since $m_i(z) = 1$ for $i > e_{\text{fin}}$ we get that there are $p^{(M+k)-(e_{\text{fin}}+k)} = p^{M-e_{\text{fin}}}$ solutions modulo p^{M+k} for all $k \in \mathbb{N}$. This shows that there is an upper bound on the number of solutions in this case. Remark that M and e_{fin} depend on z and that we can compute $M - e_{\text{fin}}$ directly as

$$M - e_{\text{fin}} = \sum_{\ell=1}^{e_{\text{fin}}} m_\ell(z) - e_{\text{fin}} = \sum_{\ell=1}^{e_{\text{fin}}} \ell(m_\ell - m_{\ell+1}) = \sum_{z_i \neq z} \nu_p(z - z_i).$$

If z is not a single root there is no such bound. Since $m_\ell(z) \leq n$ the most roots above z exist if all z_i are equal, so for $m_\ell(z) = n$ for all $\ell \in \mathbb{N}$. Then $\sum_{\ell=1}^{j} m_\ell(z) = nj$ and the number of solutions above z modulo p^e is

$$p^{e-\lceil \frac{e}{n} \rceil} \leq p^{e\frac{n-1}{n}}.$$

Up to now we only considered one fixed root z. If we want the total number of solutions of $f(x) = 0$ in $\mathbb{Z}/p^e\mathbb{Z}$ we have to be a little careful. For $\nu_p(z_i - z_j) = k$ we only distinguished between solutions above z_i and solutions above z_j modulo p^e for $e > \sum_{\ell=1}^{k} m_\ell(z_i) = \sum_{\ell=1}^{k} m_\ell(z_j)$ when we set d_k to zero. So for $\sum_{\ell=1}^{k-1} m_\ell(z_i) < e \leq \sum_{\ell=1}^{k} m_\ell(z_i)$ we say the root z_i only contributes $p^{e-k}/m_k(z_i)$ solutions modulo p^e and then we can just sum over all $i \in \{1, \ldots, n\}$. The total number of solutions modulo p^e is therefore

$$\sum_{i=1}^{n} \frac{p^{e-k_i}}{m_{k_i}(z_i)}$$

for k_i such that $\sum_{\ell=1}^{k_i-1} m_\ell(z_i) < e \leq \sum_{\ell=1}^{k_i} m_\ell(z_i)$.

So modulo p there can be at most n solutions. This requires $z_i \not\equiv z_j \bmod p$ for $i \neq j$. But the highest possible number of solutions above a given z is $p^{e\frac{n-1}{n}}$ and this requires all the z_i to be equal. Therefore the total number of solutions to $f(x) \equiv 0 \bmod p^e$ is strictly smaller than $np^{e\frac{n-1}{n}}$.

4.3 Bounding $|R|$

Since $|R|$ is the number of solutions to $a(x) \equiv 0 \bmod C$ (or $b(x) \equiv 0$ equivalently) it is sufficient to find a bound on the number of solutions per prime power

factor of C. By [9] we know that C has $\log\log C$ different prime factors, at least asymptotically. Let $C = \prod_{i=1}^{t} p_i^{k_i}$ be the prime factorisation of C. Then for each factor $p_i^{k_i}$ there are less than $np_i^{k_i\frac{n-1}{n}}$ solutions and using the Chinese Remainder Theorem the bound for the number of solutions modulo C is just the product

$$|R| < \prod_{i=1}^{t}\left(np_i^{k_i\frac{n-1}{n}}\right) = n^t\left(\prod_{i=1}^{t}p_i^{k_i}\right)^{\frac{n-1}{n}} = n^t C^{\frac{n-1}{n}} \approx n^{\log\log C} C^{\frac{n-1}{n}}. \quad (2)$$

The ratio is therefore bounded by

$$\frac{|R|}{C} \leq \frac{n^t C^{\frac{n-1}{n}}}{C} = n^t C^{-\frac{1}{n}} \approx \frac{n^{\log\log C}}{\sqrt[n]{C}} \underset{C\to\infty}{\longrightarrow} 0.$$

In practice these values are significantly smaller. For PTE solutions with $n = 7$ we find that roughly half the prime power factors of C are square-free and therefore admit at most $n = 7$ solutions in contrast to $np^{\frac{n-1}{n}}$. For these PTE solutions the bound (2) is closer to $|R|^2$ than to $|R|$. More empirical values can be found in table 2.

5 Conclusion

We have seen that the tree approach presented by Costello, Meyer and Naehrig [6] is well suited to find twin smooth integers with ideal PTE solutions of degree $n = 6$. But for primes in the range of 512 bit and above solutions of higher degree $n \geq 7$ are favourable. Since there are fewer solutions of higher degree and with small C, this new approach to only look at the relevant residues can be faster. This is for example the case for $n = 7$. Our benchmarks and the bounds on $|R|$ and $|R|/C$ led to believe that this approach is also faster for $n \geq 9$. The formula for the number of zeros of a given polynomial modulo a prime power might be of independent interest.

Computing benchmarks for $n \geq 8$ and doing a proper search over a full interval and not only for single chunks of size 2^{20} is left for future work. For such a large search parallelising the search on different chunks might be beneficial. Regarding implementations this may require adding integers larger than 128 bit in the C implementation (for $\log_2 C \geq 64$).

One major aspect of this procedure is the sieve of Eratosthenes and an improvement of this initial step would speed up the whole search. When focusing on a single PTE solution, it might be preferable to use a different approach to only check the relevant residues for smoothness and not every integer in the interval. Unfortunately no such approach could be found in this paper. As mentioned in section 3.3, this sieving process can be modified when searching for primes for SQISign.

Another interesting field are the PTE solutions themselves. On the one hand there is still the question whether there exist ideal solutions for $n = 11$ and $n \geq 13$. On the other hand there is the problem of finding solutions with small C.

Even when systematic ways to construct ideal solutions are known, they produce large differences C most of the time. Investigating this area could especially facilitate the search for larger numbers.

Acknowledgements

I would like to thank a few people, who supported me during the work on this paper: Firstly Jens Zumbrägel for his general advice and helpful comments on my work, secondly John Abbott for our discussions about implementation and finally Michael Naehrig for making the PTE solutions for $n = 7$ and $n = 8$ available.

References

1. Bernstein, D.J.: How to find Smooth Parts of Integers (2004), https://cr.yp.to/factorization/smoothparts-20040510.pdf
2. Borwein, P., Ingalls, C.: The Prouhet-Tarry-Escott problem revisited. Enseign. Math. (2) **40**(1-2), 3–27 (1994)
3. Bruno, G., Santos, M.C.R., Costello, C., Eriksen, J.K., Naehrig, M., Meyer, M., Sterner, B.: Cryptographic smooth neighbors. Cryptology ePrint Archive, Paper 2022/1439 (2022), https://eprint.iacr.org/2022/1439, https://eprint.iacr.org/2022/1439
4. Caley, T.: The Prouhet-Tarry-Escott problem. PhD thesis, University of Waterloo (2012), https://uwspace.uwaterloo.ca/bitstream/handle/10012/7205/Caley_Timothy.pdf
5. Chernick, J.: Ideal Solutions of the Tarry-Escott Problem. Amer. Math. Monthly **44**(10), 626–633 (1937), http://doi.org/10.2307/2301481
6. Costello, C., Meyer, M., Naehrig, M.: Sieving for Twin Smooth Integers with Solutions to the Prouhet-Tarry-Escott Problem. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. pp. 272–301. Springer International Publishing, Cham (2021)
7. Crandall, R., Pomerance, C.: Prime Numbers: A Computational Perspective. Springer, 2nd edn. (2005), https://doi.org/10.1007/0-387-28979-8
8. De Feo, L., Kohel, D., Leroux, A., Petit, C., Wesolowski, B.: SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020. pp. 64–93. Springer International Publishing, Cham (2020)
9. Hardy, G.H., Ramanujan, S.: The normal number of prime factors of a number n. Quart. J. **48**, 76–92 (1917)
10. Shuwen, C.: The Prouhet-Tarry-Escott Problem. Equal Sums of Like Powers (2022), http://eslpower.org/TarryPrb.htm

A Running times

n	C	$\lvert R\rvert$	target size	bound	start	running times (ms)			
						Sage		C	
						naive	new	naive	new
6	14,400	5,400	256	16	42	401	279	2.89	1.40
					43	363	251	2.71	1.32
					44	359	248	2.56	1.26
			384	24	64	382	252	2.89	1.35
					65	376	249	2.71	1.29
					66	375	247	2.58	1.25
			512	28	85	353	238	2.08	1.09
					86	358	247	2.03	1.06
					87	352	244	1.98	1.05
6	705,600	113,400	256	16	43	310	130	2.71	0.81
					44	304	130	2.55	0.80
					45	297	131	2.41	0.83
			384	24	65	364	137	2.68	0.85
					66	361	136	2.53	0.83
					67	356	135	2.47	0.79
			512	28	86	342	130	2.03	0.69
					87	342	128	1.97	0.63
					88	340	127	1.94	0.63
6	2,822,400	226,800	256	16	43	314	73.2	2.71	0.38
					44	304	72.4	2.55	0.36
					45	297	71.3	2.41	0.35
			384	24	65	368	75.5	2.68	0.52
					66	359	74.5	2.55	0.35
					67	356	74.6	2.49	0.78
			512	28	87	339	70.5	1.98	0.31
					88	342	71.0	1.93	0.30
					89	342	70.5	1.90	0.30
6	85,377,600	3,742,200	256	16	44	303	42.8	2.56	0.21
					45	297	42.1	2.41	0.20
					46	291	42.3	2.28	0.19
			384	24	66	362	43.3	2.57	0.20
					67	359	42.7	2.48	0.19
					68	351	42.1	2.38	0.19
			512	28	87	344	40.8	1.98	0.19
					88	340	40.6	1.93	0.17
					89	334	40.7	1.89	0.18
6	85,377,600	3,742,200	256	16	44	306	42.1	2.55	0.21
					45	300	41.9	2.41	0.21
					46	295	41.7	2.28	0.20
			384	24	66	361	42.3	2.59	0.21
					67	361	42.0	2.52	0.20
					68	359	42.0	2.36	0.19
			512	28	87	345	40.2	1.98	0.19
					88	345	40.4	1.93	0.18
					89	344	40.1	1.90	0.19

Table 1. Running times of the different algorithms for a search interval of size 2^{20} starting at the start value. Target size, smoothness bound and start are all in $\log_2$. More detail in section 3.4.

PTE solution				target	smooth.	start	running time (ms)	
parameters		sizes		size	bound		naive	new
C	5,145,940,800	C	33 bit	256	17	40	2.436	0.089
$\|R\|$	101,879,232	$\|R\|$	27 bit	384	25	60	2.056	0.073
ratio	$1.98 \cdot 10^{-2}$	R	0.8 GB	512	27	80	1.024	0.043
C	13,967,553,600	C	34 bit	256	17	40	2.556	0.075
$\|R\|$	238,592,172	$\|R\|$	28 bit	384	25	60	2.071	0.082
ratio	$1.71 \cdot 10^{-2}$	R	1.8 GB	512	27	80	1.024	0.043
C	293,318,625,600	C	39 bit	256	17	40	2.265	0.033
$\|R\|$	1,845,811,968	$\|R\|$	31 bit	384	25	60	0.837	0.022
ratio	$6.29 \cdot 10^{-3}$	R	14 GB	512	27	80	0.691	0.021
C	1,037,896,675,200	C	40 bit	256	17	40	2.553	0.020
$\|R\|$	3,017,192,640	$\|R\|$	32 bit	384	25	60	0.865	0.011
ratio	$2.91 \cdot 10^{-3}$	R	22 GB	512	27	80	0.717	0.010
C	5,771,633,313,600	C	43 bit	256	17	40	2.807	0.007
$\|R\|$	1,670,145,204	$\|R\|$	31 bit	384	25	60	0.866	0.005
ratio	$2.89 \cdot 10^{-4}$	R	12 GB	512	27	80	0.714	0.005
C	1,440,534,083,788,800	C	51 bit	256	17	40		
$\|R\|$	4,753,277,448,192	$\|R\|$	43 bit	384	25	60		
ratio	$3.30 \cdot 10^{-3}$	R	35 TB	512	27	80		
C	11,471,328,290,822,400	C	54 bit	256	17	40		
$\|R\|$	176,264,555,376	$\|R\|$	38 bit	384	25	60		
ratio	$1.54 \cdot 10^{-5}$	R	1.3 TB	512	27	80		
C	15,256,916,548,473,600	C	54 bit	256	17	40		
$\|R\|$	113,569,873,872	$\|R\|$	37 bit	384	25	60		
ratio	$7.44 \cdot 10^{-6}$	R	0.8 TB	512	27	80		
C	106,911,286,818,336,000	C	57 bit	256	17	40		
$\|R\|$	1,307,456,866,800	$\|R\|$	41 bit	384	25	60		
ratio	$1.22 \cdot 10^{-5}$	R	10 TB	512	27	80		
C	314,073,647,406,288,000	C	59 bit	256	17	40		
$\|R\|$	457,609,903,380	$\|R\|$	39 bit	384	25	60		
ratio	$1.46 \cdot 10^{-6}$	R	3.3 TB	512	27	80		

Table 2. Running times of the different algorithms implemented in C for a search interval of size 2^{20} starting at the start value. Target size, smoothness bound and start are all in $\log_2$. All PTE solutions have $n = 7$. More detail in section 3.4.