A low-memory parallel version of Matsuo, Chao and Tsujii's algorithm

Pierrick Gaudry, Eric Schost

A low-memory parallel version of Matsuo, Chao and Tsujii's algorithm

Pierrick Gaudry[1] and Éric Schost[2]

[1] Laboratoire LIX, École polytechnique, 91128 Palaiseau, France
gaudry@lix.polytechnique.fr
[2] Laboratoire STIX, École polytechnique, 91128 Palaiseau, France
Eric.Schost@polytechnique.fr

Abstract. We present an algorithm based on the birthday paradox, which is a low-memory parallel counterpart to the algorithm of Matsuo, Chao and Tsujii. This algorithm computes the group order of the Jacobian of a genus 2 curve over a finite field for which the characteristic polynomial of the Frobenius endomorphism is known modulo some integer. The main tool is a 2-dimensional pseudo-random walk that allows to heuristically choose random elements in a 2-dimensional space. We analyze the expected running time based on heuristics that we validate by computer experiments. Compared with the original algorithm by Matsuo, Chao and Tsujii, we lose a factor of about 3 in running time, but the memory requirement drops from several GB to almost nothing. Our method is general and can be applied in other contexts to transform a baby-step giant-step approach into a low memory algorithm.

1 Introduction

Jacobians of small genus curves have now become an important tool for public-key cryptography; computing the Zeta function of such curves remains one of the central problems in this area.

For elliptic curves, the question found a first answer with Schoof's algorithm and the subsequent improvements (see [4] and the references therein). Furthermore, if the characteristic of the definition field is small, p-adic methods initiated with Satoh's algorithm [18] give a tremendous speed-up.

For higher genus curves, the p-adic methods (based on either Mestre's [15] or Kedlaya's [12] algorithms) also yield very satisfactory solutions in small characteristic, both in theory and in practice. However, in large characteristic, the question remains delicate. From the theoretical point of view, Pila's algorithm [16] and subsequent improvements [2, 10] give polynomial time solutions, following Schoof's strategy. However, these have been turned into practical algorithms for genus 2 curves only [8], and it was only very recently that a Jacobian of cryptographic size was counted that way [9].

In this paper, we concentrate on the last part of a Schoof-like genus 2 algorithm. We first recall the basics of such algorithms.

Let us denote by $\chi \in \mathbb{Z}[T]$ the characteristic polynomial of the Frobenius endomorphism, so that $\chi(1)$ equals the Jacobian order. A basic task is to compute the reduction of χ modulo ℓ, for sufficiently many small primes or prime powers ℓ. Such information is obtained through the study of the torsion in the Jacobian; if the characteristic is medium, the Cartier-Manin operator [8, 6] can also be used.

Thanks to Weil's bounds, one can then collect modular information until χ is known. However, it is far better to switch to a baby-step giant-step (BSGS) algorithm before the end, since collecting modular information is costly. For instance, in [9], the highest prime ℓ was 19, whereas without the BSGS phase, it would have been 59.

If $\chi(1)$ is known modulo m, then using standard BSGS techniques, the time necessary to compute the Jacobian order varies like $1/\sqrt{m}$. In 2002, Matsuo, Chao and Tsujii [14] showed that if we use not only the value $\chi(1)$ modulo m, but rather all coefficients of χ modulo m, it is possible to speed-up the BSGS computation. The runtime of their method (called MCT in the following) varies like $1/m$, which is an important improvement. The main drawback is the space complexity: the largest example shown in [14] used 12 GB of central memory, whereas the runtime (5 days on a single processor) was quite reasonable.

Standard BSGS techniques have low-memory, parallelizable, probabilistic counterparts, based on the rho or lambda (kangaroo) methods of Pollard's [17]: such techniques were presented in [24, 22, 23, 8, 21]. In this paper, we apply the ideas of Matsuo *et al.* to improve the variant of [8]; we obtain a probabilistic algorithm, with a heuristic complexity analysis, but which requires almost no memory and is immediately parallelizable. The expected running time is about 3 times the running time of the MCT algorithm.

In order to simplify the exposition, we concentrate on genus 2 curves. However, our idea works for more general curves as soon as the characteristic polynomial χ is known modulo some integer m. Another application of our method is the algorithm of [3] for Picard curves, which is a BSGS type algorithm.

The paper is organized as follows. Section 2 defines the necessary notation and recalls the BSGS algorithm of [14]. Then, to introduce methods based on the birthday paradox, we start in Section 3 by a special case, where the modular information is complete enough, so we can use the method of [8]. In Section 4, we deal with the general case, and introduce bidimensional analogues of these techniques. Section 5 finally presents our experimental results.

2 MCT algorithm for genus 2 hyperelliptic curves

2.1 Characteristic polynomial of Frobenius endomorphism

Let $\mathcal{C}$ be a genus 2 curve, defined over the finite field $\mathbb{F}_q$ with q elements. The Jacobian group of $\mathcal{C}$ is denoted by $\mathbf{J}(\mathcal{C})$ and the characteristic polynomial of the Frobenius endomorphism is denoted by $\chi(T)$. This polynomial has the form

$$\chi(T) = T^4 - s_1 T^3 + s_2 T^2 - q s_1 T + q^2,$$

where s_1 and s_2 are integers. The group order of the Jacobian is then given by

$$\#\mathbf{J}(\mathcal{C}) = \chi(1) = q^2 + 1 - s_1(q+1) + s_2.$$

By Weil's theorem, the roots of χ have absolute value $\sqrt{q}$ and bounds on s_1 and s_2 follow directly. Better bounds can be found using the fact that the roots of χ come in pairs of complex conjugates [13, Proposition 7.1]:

$$|s_1| \leq 4\sqrt{q}, \qquad 2|s_1|\sqrt{q} - 2q \leq s_2 \leq \frac{s_1^2}{4} + 2q. \tag{1}$$

The values of (s_1, s_2) satisfying these bounds form the hatched zone in Figure 1.

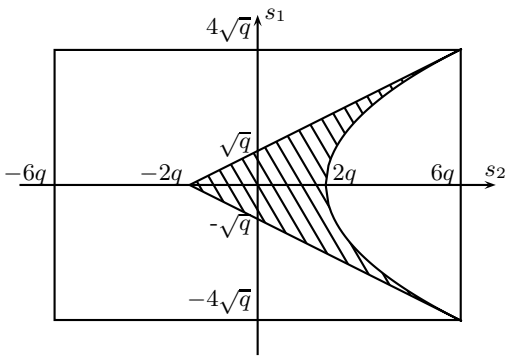

Fig. 1. Bounds on s_1 and s_2

2.2 Review of the MCT algorithm

In large characteristic, point counting algorithms work by collecting modular information; this information is then recombined using the Chinese Remainder Theorem and the work is finished using some BSGS strategy. We now describe the MCT algorithm for this last step.

From now on, we assume that the characteristic polynomial χ is known modulo some positive integer m, *i.e.* s_1 and s_2 are known modulo m. Therefore we introduce new variables for the known and unknown parts of s_1 and s_2:

$$s_1 = \overline{s_1} + m\widetilde{s_1} \quad \text{and} \quad s_2 = \overline{s_2} + m\widetilde{s_2},$$

so that our goal is now to find $\widetilde{s_1}$ and $\widetilde{s_2}$. To this effect, a random divisor D is first picked in $\mathbf{J}(\mathcal{C})$: the strategy is to compute the order of D, hoping that it is large enough to be able to conclude (the case when $\mathbf{J}(\mathcal{C})$ is highly non-cyclic is rare in practice and easily tackled).

The order of D divides the group order $\chi(1)$, therefore we have the equality

$$(q^2 + 1 - \overline{s_1}(q+1) + \overline{s_2}) \cdot D + (-\widetilde{s_1}(q+1) + \widetilde{s_2}) \cdot m \cdot D = 0.$$

To obtain a BSGS algorithm, it is usual to separate the two unknowns, one on each side of the equation. Here the unknowns $\widetilde{s_1}, \widetilde{s_2}$ lie in intervals that are of different sizes, therefore it is necessary to split again $\widetilde{s_2}$ into two pieces. Let n be a parameter to be fixed later, and let us write $\widetilde{s_2} = t_2 + nu_2$ with $0 \leq t_2 < n$. Then $\chi(1) \cdot D = 0$ rewrites

$$\left(q^2 + 1 - \overline{s_1}(q+1) + \overline{s_2} + m(-\widetilde{s_1}(q+1) + nu_2)\right) \cdot D = -t_2 m \cdot D.$$

The algorithm proceeds as follows: first, all possible values for the right-hand side are computed and stored in a data-structure in which searching is fast. Then the left-hand side is computed for all possible values of $\widetilde{s_1}$ and u_2, until a match is found with a value of the right-hand side. For each value of $\widetilde{s_1}$, the bounds of Equation (1) are used to find the range of the possible values for u_2; a precise study of the area of the space of search leads to an optimal value $n \approx q^{3/4}/m$, yielding a running time of about $4q^{3/4}/m$ operations in $\mathbf{J}(\mathcal{C})$.

3 Preliminaries: the case $m \geq 8\sqrt{q}$

The main drawback of the previous method is the storage requirement. We now introduce low-memory variants and start by describing a special case, for which most features of the general treatment are already present.

Suppose that χ is known modulo m, with $m \geq 8\sqrt{q}$. Because $|s_1| \leq 4\sqrt{q} \leq m/2$, the coefficient s_1 is known exactly. Corresponding to this value of s_1, we have bounds for s_2 which yield bounds on $\widetilde{s_2}$, which is the only remaining unknown. We are thus in the setting of the search of a group order in a bounded arithmetic progression. Among the many variants inspired by Pollard's lambda (or kangaroos) method that yield a low-memory solution to this question, we chose the one based on the birthday paradox described in [8]. We recall it here briefly; detailed descriptions and analyses of several other variants can be found in [17, 24, 22, 23, 21].

3.1 Random search in two intersecting intervals

Let $K = q^2 + 1 - s_1(q+1) + \overline{s_2}$. Then the group order can be written $\#\mathbf{J}(\mathcal{C}) = K + m\widetilde{s_2}$, where K is known and $\widetilde{s_2}$ is unknown. Performing a suitable shift, it is possible to adjust K by some multiple of m so that the bounds on $\widetilde{s_2}$ become symmetric, *i.e.* such that $|\widetilde{s_2}| \leq B$ for some integer B.

As before, we pick at random an element D in $\mathbf{J}(\mathcal{C})$, of presumably large order. We then define two sets of divisors that we denote W and T:

$$W = \{(K + m\sigma_2) \cdot D \ ; \ \sigma_2 \in [-B, B]\}, \quad T = \{m\sigma_2 \cdot D \ ; \ \sigma_2 \in [-B, B]\}.$$

By definition W and T intersect, since the zero divisor is in both of them, by taking $\sigma_2 = \widetilde{s_2}$ for W and $\sigma_2 = 0$ for T. More precisely, the size of the intersection is $\#(W \cap T) \in [B+1, 2B+1]$, depending on how far $\widetilde{s_2}$ is from 0. The algorithm then proceeds as follows: we pick random elements uniformly alternatively in W

and T. If the same divisor is obtained as an element of both W and T then (a multiple of) the order of D can be deduced.

Assuming that picking a random element in W or T has unit cost, that all elements are stored in table, and that a search in the table has unit cost, then by the birthday paradox, a collision can be obtained in expected time and space $O(\sqrt{B})$. The constant in the $O(\ )$ can actually be made more explicit. First, for fixed $\widetilde{s_2}$, the expected running time grows like $\sqrt{(2B+1)\pi/\gamma}$, where $\gamma = \#(W \cap T)/(2B+1) = (2B+1-|\widetilde{s_2}|)/(2B+1)$. Thus, assuming that $\widetilde{s_2}$ is uniformly distributed in $[-B, B]$, the expected number of operations is asymptotic to

$$\frac{1}{2B+1}\int_{-B}^{B}\sqrt{\frac{(2B+1)\pi}{\gamma(\widetilde{s_2})}}\,d\widetilde{s_2} = \int_{-B}^{B}\sqrt{\frac{\pi}{2B+1-|\widetilde{s_2}|}}\,d\widetilde{s_2},$$

which itself is asymptotic to $2(2-\sqrt{2})\sqrt{2\pi B} \approx 2.07\sqrt{2B}$. Though, one should note that even if $\mathcal{C}$ is randomly chosen uniformly among all curves, the number of points of the Jacobian is not uniformly distributed, as the concentration is slightly higher in the middle of the Hasse-Weil interval; therefore $\widetilde{s_2}$ is not uniformly distributed.

3.2 Pseudo-random walk and distinguished points

We now address the main issues raised in the above algorithm: the generation of random elements in W and T and their storage. The key to the answer is to replace randomness by a deterministic pseudo-random walk.

Let $r \geq 0$ and ℓ be parameters to be fixed later. The pseudo-random walk is initialized as follows: for all k in $[1, r]$, an offset $\mathcal{O}_k \in \mathbf{J}(\mathcal{C})$ is precomputed as $\mathcal{O}_k = \alpha_k m \cdot D$, where α_k is a random integer in $[0, 2\ell]$. We also need a hash function $\mathcal{H}$ that maps elements of $\mathbf{J}(\mathcal{C})$ to $[1, r]$; this hash function should have good statistical properties, but no cryptographically strong property, like one-wayness, is required. Typically, $\mathcal{H}$ is obtained by taking a few bits in the internal representation of the elements and the integer obtained this way is taken modulo r. Then, starting with an element P in W (resp. in T) for which we know the corresponding σ_2, we define another point Q by $Q = P + \mathcal{O}_{\mathcal{H}(P)}$.

Assuming that ℓ is not too large compared to B, with high probability the point Q is still an element of W (resp. of T). Furthermore, the value of σ_2 corresponding to Q is obtained by adding $\alpha_{\mathcal{H}(P)}$ to the value of σ_2 for P.

Iterating this process yields a chain of pseudo-randomly chosen elements in W (resp. in T). However, the chain should not be too long, to keep the probability to go out of the domain moderate. Thus the average length ℓ of an offset must be adapted according to the average number of steps we expect to do in one chain (see below). With this device it is then possible to produce each new pseudo-random element in W or T for one operation in $\mathbf{J}(\mathcal{C})$.

We now deal with storage requirements. To this effect, we introduce the concept of distinguished points, originally appeared in [7]. We say that an element

of $\mathbf{J}(\mathcal{C})$ is a distinguished point if its image by a second hash function is 0. Again, we do not ask much of this hash function, except that its behavior looks independent of any arithmetic property of the element, and that the probability of being distinguished can be effectively estimated and tuned to a prescribed value. As usual, looking at some bits in the internal representation of the elements is a good way of doing. We denote by $p_{\mathcal{D}}$ the probability for an element of being distinguished.

The algorithm then proceeds as follows: starting with a random point alternatively in W and T, we produce pseudo-random elements using the pseudo-random walk, until a distinguished point is hit. Then this point is stored and another chain is started. The length of the chains is $1/p_{\mathcal{D}}$ on average, and the parameter ℓ should be tuned accordingly. If all the parameters are well chosen, then after (say) 1000 chains on average, we have produced enough points to expect a collision. If this occurs at a point that is not distinguished, then the two chains continue on the same track because the pseudo-random walk is deterministic; the two chains will end at the same distinguished point, thus allowing the detection and the solution of the problem.

Many practical experiments were made to test this strategy against the idealized algorithm described above. They are very satisfactory and in [22] it was suggested that taking $r = 20$ is enough to simulate a random walk in this context.

As a conclusion, in the case when $m \geq 8\sqrt{q}$, the number of points can be computed using a parallel low-memory algorithm that requires on average $O(\sqrt{q/m})$ operations in $\mathbf{J}(\mathcal{C})$. This is worse than the $O(q^{3/4}/m)$ complexity announced in [14], but their complexity analysis implicitly excludes that case.

4 The general case

In the case when $m < 8\sqrt{q}$, there are several choices for s_1 and still many more for s_2. We could loop over all the possible values of s_1 and look for a corresponding s_2 using the algorithm of the previous section, but this approach has worse complexity than the MCT algorithm. The workaround is to take into account the bidimensional nature of the problem.

Recall that we have $s_1 = \overline{s_1} + m\widetilde{s_1}$ and $s_2 = \overline{s_2} + m\widetilde{s_2}$, where $\overline{s_1}$ and $\overline{s_2}$ are known integers in $[0, m-1]$, the goal being to find $\widetilde{s_1}$ and $\widetilde{s_2}$. From the bounds (1) on s_1 and s_2, we deduce similar bounds for $\widetilde{s_1}$ and $\widetilde{s_2}$. In order to make the description more generic, we write these bounds in the following form:

$$B_{1,\min} \leq \widetilde{s_1} \leq B_{1,\max}, \qquad B_{2,\min} \leq \widetilde{s_2} \leq B_{2,\max}.$$

4.1 Random search in two intersecting rectangles

Let D be a random element of $\mathbf{J}(\mathcal{C})$; as previously, we assume that the order of D is large enough compared to the group order. Since $\chi(1) \cdot D = 0$, we have

$$\left(q^2 + 1 - \overline{s_1}(q+1) + \overline{s_2}\right) \cdot D + (-\widetilde{s_1}(q+1) + \widetilde{s_2}) \cdot m \cdot D = 0.$$

Let K be $q^2+1-\overline{s_1}(q+1)+\overline{s_2}$, which is a known integer; then we have to find $\widetilde{s_1}$ and $\widetilde{s_2}$ such that $K \cdot D + (-\widetilde{s_1}(q+1)+\widetilde{s_2})\, m \cdot D = 0$. Let R be a rectangle containing the possible values for the pair $(\widetilde{s_1}, \widetilde{s_2})$:

$$R = \{(\sigma_1, \sigma_2);\ \sigma_1 \in [B_{1,\min}, B_{1,\max}],\ \sigma_2 \in [B_{2,\min}, B_{2,\max}]\}\,.$$

Since $B_{1,\min}$ (resp. $B_{2,\min}$) is not necessarily the opposite of $B_{1,\max}$ (resp. $B_{2,\max}$), we have to normalize the situation. We define a value K' to be used in place of K, so as to center our search:

$$K' = K + m\left(-\left\lfloor \frac{B_{1,\min}+B_{1,\max}}{2}\right\rfloor (q+1) + \left\lfloor \frac{B_{2,\min}+B_{2,\max}}{2}\right\rfloor\right).$$

We can now define two sets of points:

$$W = \{K' \cdot D + (-\sigma_1(q+1)+\sigma_2)\cdot m \cdot D;\ (\sigma_1,\sigma_2) \in R\}\,,$$

$$T = \{(-\sigma_1(q+1)+\sigma_2)\cdot m \cdot D;\ (\sigma_1,\sigma_2) \in R\}\,.$$

We assume that these two sets have cardinality exactly $\#R$. This may not hold in general, but is true with the further assumptions that D is of large order and that m is larger than 8.

By construction, the sets W and T have a non-trivial intersection. Let D_W be in W and D_T in T. We write $(\sigma_{1W}, \sigma_{2W})$ the values corresponding to D_W and $(\sigma_{1T}, \sigma_{2T})$ the values corresponding to D_T. Then, assuming again that the order of D is large enough, $D_W = D_T$ if and only if

$$\begin{aligned} \sigma_{1W} - \sigma_{1T} &= \widetilde{s_1} - \lfloor (B_{1,\min}+B_{1,\max})/2 \rfloor\,, \\ \sigma_{2W} - \sigma_{2T} &= \widetilde{s_2} - \lfloor (B_{2,\min}+B_{2,\max})/2 \rfloor\,. \end{aligned} \tag{2}$$

Hence it is easily checked that

$$N = \#(W \cap T) \in \left[\frac{\#R}{4}, \#R\right].$$

When $B_{1,\min} = -B_{1,\max}$ and $B_{2,\min} = -B_{2,\max}$, we get the picture of Figure 2.

A first version of our algorithm now proceeds as follows: random elements of W and T are constructed by picking random elements in R. These elements are stored in a data structure in which it is possible to detect quickly collisions between an element of W and an element of T. Together with these elements, we also store the corresponding pair (σ_1, σ_2). On average, due to the birthday paradox, a collision occurs after having constructed $O(\sqrt{N})$ elements of W and T; taking the difference of the pairs σ_T and σ_W then gives the result by Equations (2).

Since the bounds on $\widetilde{s_1}$ and $\widetilde{s_2}$ yield $|\widetilde{s_1}| = O(\sqrt{q}/m)$ and $|\widetilde{s_2}| = O(q/m)$, we have $N = O(q^{3/2}/m^2)$, and therefore the expected number of points to construct is in $O(q^{3/4}/m)$. Just as in the unidimensional case, we now give an estimate of the constant hidden in the $O(\ )$, using simplifying assumptions.

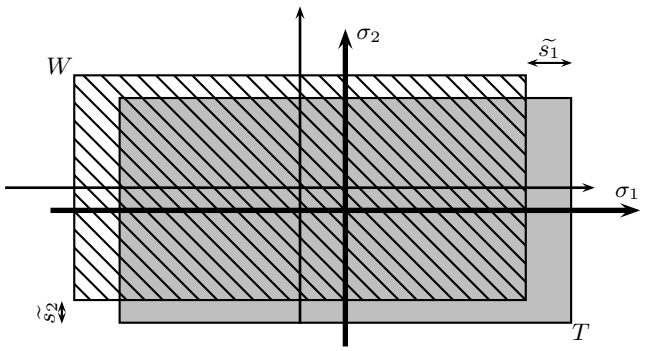

Fig. 2. Intersection of W and T

For such estimates, we can assume that the problem is centered, and therefore we put $B_1 = B_{1,\max} = -B_{1,\min}$ and $B_2 = B_{2,\max} = -B_{2,\min}$; hence $\#R = (2B_1 + 1)(2B_2 + 1)$. We denote by $\gamma \in [\frac{1}{4}, 1]$ the ratio $\#(W \cap T)/\#R$: this parameter is easily computed as:

$$\gamma(\widetilde{s_1}, \widetilde{s_2}) = \frac{(2B_1 + 1 - |\widetilde{s_1}|)(2B_2 + 1 - |\widetilde{s_2}|)}{\#R}.$$

From the birthday paradox, we see that the expected number of elements that have to be created before a collision between an element of W and an element of T occurs is asymptotic to $\sqrt{\pi\,\#R/\gamma}$.

We now assume that $\widetilde{s_1}$ and $\widetilde{s_2}$ are uniformly distributed. Then the average number of points to construct grows like

$$\frac{1}{\#R} \int_{\widetilde{s_1}=-B_1}^{B_1} \int_{\widetilde{s_2}=-B_2}^{B_2} \sqrt{\frac{\pi\,\#R}{\gamma(\widetilde{s_1}, \widetilde{s_2})}}\; d\widetilde{s_1}\, d\widetilde{s_2}.$$

This integral is easily computed and shows that the expected number of points to construct is asymptotic to $8\sqrt{\pi}(3-2\sqrt{2})\sqrt{\#R} \approx 2.43\,\sqrt{\#R}$. Now, bounds (1) on s_1 and s_2 yield approximate bounds for $\widetilde{s_1}$ and $\widetilde{s_2}$:

$$B_{1,\max} - B_{1,\min} = 8\sqrt{q}/m,\ B_{2,\max} - B_{2,\min} = 8q/m,$$

hence $\#R = 64\,q^{3/2}/m^2$. The approximate value of 2.43 then yields a running time of about $19.5\,q^{3/4}/m$ operations in $\mathbf{J}(\mathcal{C})$. Hence, we see that a constant factor of about 5 is lost compared to the original (memory consuming) MCT algorithm. This difference is partly due to the fact that in the MCT algorithm, the BSGS approach allows to search only in the area described in Figure 1, whereas ours does not take this specificity into account.

Note that this analysis is idealized in several places: first, the pseudo-random walk that will be used below is not purely random and can cause some discrepancies. Next, the assumption that $\widetilde{s_1}$ and $\widetilde{s_2}$ are uniformly distributed in a rectangle is actually wrong. However, we expect that this should still give a good estimate.

4.2 A bidimensional pseudo-random walk

The main questions are now the generation of random elements in W and T, and how to avoid storing them. These are the same issues as in Section 3 and similar techniques will be applied to solve them. The definition of distinguished points still makes sense in this new context, and will be used with no modification. However, the previous pseudo-random walk needs to be converted into a bidimensional one.

Let r, ℓ_1 and ℓ_2 be parameters to be fixed later: r controls the number of offsets that we are going to precompute, and ℓ_1 and ℓ_2 are the average lengths of the horizontal and vertical offsets. For each k and k' in $[1, r]$, we select a random non-negative integer $\alpha_{k,k'}$ uniformly in $[0, 2\ell_1]$ and a random non-negative integer $\beta_{k,k'}$ uniformly in $[0, 2\ell_2]$. Then for each k and k' in $[1, r]$ and b in $\{0, 1\}$, we compute and store the offsets $\mathcal{O}_{k,k',b} = (-1)^b \alpha_{k,k'}(q+1)mD + \beta_{k,k'}mD \in \mathbf{J}(\mathcal{C})$, where D is the base point whose order is to be computed.

Starting with a point P in W (resp. in T) for which we know the corresponding pair (σ_1, σ_2), we define another point as follows. We compute k, k', and b as pseudo-random deterministic functions of P, by using some hash functions. Then we define $Q = P + \mathcal{O}_{k,k',b}$. If ℓ_1 and ℓ_2 are small enough, with high probability Q is still in W (resp. in T) and the corresponding pair is given by $(\sigma_1 - (-1)^b \alpha_{k,k'}, \sigma_2 + \beta_{k,k'})$. Iterating this process allows us to produce chains of pseudo-randomly chosen elements in W (resp. in T). As before, the chains can not be too long, otherwise they go out of W (resp. of T). The cost of producing one element is one group operation.

Note that we have only used positive values for the offset in the second direction: this is intended to reduce the chance of creating cycles. However, experiments with alternative strategies turned out to yield similar results.

4.3 Setting the parameters

Let λ be such that the expected number of points to construct is $\lambda\sqrt{\#R}$ and let C be the number of chains we expect to construct. Note that C is fixed by the user; it should be large enough so that averaging considerations make sense (say $C \geq 1000$), and small enough so that the cost of initializing a chain is negligible compared to the cost of the steps that are done in that chain. Also the number of chains is essentially the number of distinguished points that have to be stored and therefore should be small enough. Knowing an approximation of λ and having fixed C, the probability of being distinguished follows from

$$p_{\mathcal{D}} = \frac{C}{\lambda\sqrt{\#R}}.$$

Now we fix ℓ_1 and ℓ_2; to control the probability of going out of W (or T), we first evaluate the average length of a chain. There are about $1/p_{\mathcal{D}}$ steps and each step goes on average ℓ_1 in the first direction and ℓ_2 in the second direction. In the second direction, all the offsets are positive and therefore the length of

a chain is $\ell_2/p_{\mathcal{D}}$ on average. We want this to be small enough compared to the size of R in that direction, say one tenth of $B_{2,\max} - B_{2,\min}$:

$$\ell_2 = \frac{(B_{2,\max} - B_{2,\min})\, p_{\mathcal{D}}}{10}.$$

In the first direction, the situation is different since the offsets can be positive or negative, but still with an average absolute value of ℓ_1. The central limit theorem applied to the 1-dimensional random walks gives that the average distance to the origin after $1/p_{\mathcal{D}}$ steps is about $2\sqrt{2/3\pi}\,\ell_1/\sqrt{p_{\mathcal{D}}}$: the factor $2/\sqrt{3}$ corresponds to the standard deviation of the lengths of the offsets. For convenience, we approximate $2\sqrt{2/3\pi}$ by $9/10$. Again, imposing that this value is one tenth of the size of R in that direction yields:

$$\ell_1 = \frac{(B_{1,\max} - B_{1,\min})\,\sqrt{p_{\mathcal{D}}}}{9}.$$

It may happen that ℓ_1 or ℓ_2 is very small, and even smaller than 1. This is especially the case when the bounds on $\widetilde{s_1}$ and $\widetilde{s_2}$ have a different order of magnitude. Unfortunately this is the case in the genus 2 point counting case, where $\widetilde{s_2}$ is on average about $\sqrt{q}$ times larger than $\widetilde{s_1}$. A solution would be to enlarge C, but this is not satisfactory since it implies more storage.

A better choice is to modify the random walk as follows. Assume that ℓ_1 is small (ℓ_1 and ℓ_2 can not be simultaneously small). Then with probability p, we add either the same offset $\mathcal{O}_{k,k',b}$ as before or a modified offset $\widetilde{\mathcal{O}}_{k,k'} = \beta_{k,k'} mD$ which does not include a progression in the first direction. The probability p is fixed so that the apparent mean value of ℓ_1 is the one we wanted. Note that the decision of adding $\mathcal{O}_{k,k',b}$ or $\widetilde{\mathcal{O}}_{k,k'}$ is a deterministic choice that depends on D.

4.4 Reducing the search space

In order to have a better understanding of the distribution of (s_1, s_2), we ran some statistics. Note that similar statistics, supported by theoretical and heuristic considerations were done in [20], but with the purpose of finding the mean value of the class number.

Here, for $p = 10^6 + 3$, we randomly selected $10,000$ monic squarefree polynomials of degree 5 over $\mathbb{F}_p$ and computed the (s_1, s_2) values for the corresponding curves. As expected, the pairs (s_1, s_2) tend to be not too close to the borders of the domain. In Figure 3, we represented the domain where (s_1, s_2) are allowed to stay according to bounds (1), and inside the domain, the darkness of a point means that the density of pairs (s_1, s_2) is high.

On the picture, we see that there are very few pairs for which s_2 is large, because the "wings" are very thin. In fact, these points correspond to curves which are close to maximal curves, and it is no surprise that they are rare. More precisely, in our tests, the proportion of curves for which s_2 is larger than $3q$ is about 2.2% and the proportion of curves for which s_2 is larger than $4q$ is about 0.23% (remember that in theory, s_2 can be as large as $6q$).

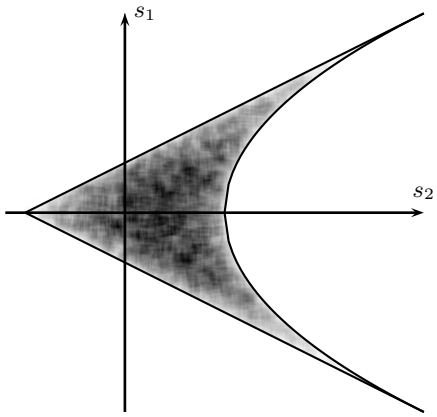

Fig. 3. Statistics on (s_1, s_2)

In view of these results, in order to improve our constant 19.5, and remarking that there is no point in spending too much time for large values of s_2, we will restrict the rectangle R to the following bounds:

$$R = \{(\sigma_1, \sigma_2);\ \sigma_1 \in [-2.5\sqrt{q}/m, 2.5\sqrt{q}/m],\ \sigma_2 \in [-2q/m, 3q/m]\}\,.$$

In our statistics, for more than 97% of the curves, $\widetilde{s_2} < 3q/m$, and therefore the overlapping factor of W and T is at least $1/4$; and for about 99.7% of the curves, $\widetilde{s_2} < 4q/m$ and therefore the overlapping factor is at least $\frac{3}{25} = 0.12$. The overlapping decreases as the point (s_1, s_2) gets closer to the end of the "wings" of the arrow on the picture and if $s_2 > 5.5\,q$, the sets W and T do not overlap.

With this strategy, the area of R is reduced to $25\,q^{3/2}/m^2$ so that the expected runtime is about $12\,q^{3/4}/m$ group operations. Therefore we lose "only" a factor of 3 compared to the original MCT algorithm; this strategy is used in the experiments presented below.

We expect that this is very unlikely to get curves with s_1, s_2 outside of the above rectangle R by random constructions. In case the algorithm does not find an answer after say 10 times the above expected time it could pay to start a classical MCT algorithm to search deterministically in the ends of the wings if we have enough memory for this much smaller subproblem. Otherwise, another method is to start a chain corresponding to these wings in the area outside the rectangle with a small probability, so that we do not perturb the average runtime but we can guarantee that the program finishes.

5 Practical experiments

We did two kinds of experiments: first with a high level implementation using the Magma computer algebra system [5], we ran some simulations to check the

validity of our approach in various situations. Then, we wrote a parallel C++ implementation of our algorithm using the NTL [19] and the MPICH [1] libraries, in order to run tests with real sized curves. In our experiments, we used the reduced search space given in Section 4.4, with $\#R \approx 25\, q^{3/2}/m^2$.

5.1 Simulations

In order to test the validity of our heuristics on sufficiently many examples of reasonable size, we used the following simulated algorithm. Let p be an integer. We pick two integers s_1 and s_2 at random uniformly in the area of Figure 1. Then we form the integer $N = p^2 + 1 - s_1(p+1) + s_2$ and work in the group $\mathbb{Z}/N\mathbb{Z}$ instead of $\mathbf{J}(\mathcal{C})$. An integer m is chosen, and using only p and s_1, s_2 modulo m, our goal is to recover N. Thus, we can construct appropriate examples with different values for the parameters at almost no cost.

We chose several pairs (p, m) yielding always $\#R \approx 5 \times 10^{10}$, so that $p_{\mathcal{D}}$ can be taken to be 2^{-8}, and with ℓ_1 varying from 0.01 to 21. For each such pair (p, m), 100 random pairs (s_1, s_2) where tested and the average number of steps is measured and compared to $p^{3/4}/m$. The results are reported in Table 1.

p	m	ℓ_1	ℓ_2	Avg ratio nb jumps / $(p^{3/4}/m)$
5.7×10^7	14	21	8,040	12.7
9.2×10^8	118	10	15,250	12.7
1.5×10^{10}	949	5	30,350	12.5
1.5×10^{14}	953674	0.5	304,000	12.6
1.2×10^{17}	144675925	0.1	1,621,271	14.3
1.9×10^{18}	1157407407	0.05	3,242,542	14.5
2.5×10^{21}	250000000000	0.01	19,455,252	22.2

Table 1. Simulations with cyclic groups $\mathbb{Z}/N\mathbb{Z}$. Each line corresponds to 100 runs.

Our conclusion is that even if the rectangle R is very thin (*i.e.* ℓ_1 is tiny), then the measured running time is quite close to the predictions. In the extreme, the last case $\ell = 0.01$ corresponds to a case where $m \approx 4\sqrt{q}$, therefore $B_{1,\max} - B_{1,\min}$ is just 2. In that case, the heuristics we used hardly make sense, but this is a good surprise that the average running time is still within a factor 2 of the heuristic analysis.

5.2 Several runs on the same curve

We ran our software implementation many times on a given curve, in order to check that the average measured running time is close to the heuristic estimate, for a medium sized problem.

Let $p = 5 \times 10^{24} + 8503491$ and f a random monic squarefree polynomial of degree 5 over $\mathbb{F}_p$. Using the Schoof-like method described in [9] we have deduced the values of s_1 and s_2 modulo $m = 44696171520$. The curve is such that $s_1/\sqrt{p} \approx -0.84$ and $s_2/p \approx 0.38$, therefore it is not close to any border of the bounds.

With the MCT algorithm, the computation is feasible and requires about 1.5 GB of memory. We ran our software 100 times on that input, and for each run we measured the number of operations in the Jacobian. The average value is $11.27\,p^{3/4}/m$, which is in accordance with our estimates. The minimal value is $2.69\,p^{3/4}/m$ and the maximal value is $31.55\,p^{3/4}/m$. In Figure 4 we give the histogram for the number of runs whose running time is in a given range.

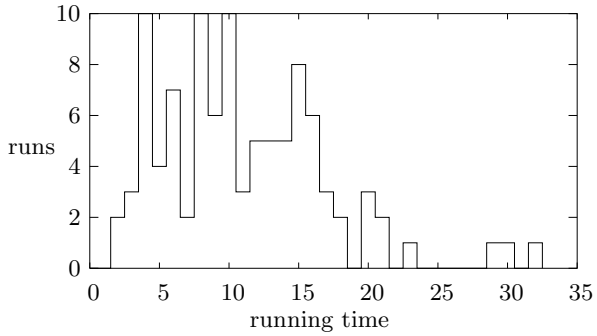

Fig. 4. Histogram showing the number of runs whose running time is in the given range (divided by $p^{3/4}/m$).

5.3 A larger example

In order to test the scalability of our method, we ran a larger example. Let $p = 5 \times 10^{24} + 8503491$ as above, and f a random monic squarefree polynomial of degree 5 over $\mathbb{F}_p$. We now suppose that s_1 and s_2 are known modulo only $m = 1655413760$ (note that the modulus 44696171520 mentioned above equals 27×1655413760).

With this more restricted information, we cannot conclude using the original MCT algorithm: the computation would require to store about 5×10^9 points together with their indices. Even using hash tables, it seems difficult to use less than 8 bytes for each entry, which means at least 40 GB of memory.

We ran our algorithm on that input. The probability of being distinguished was set to $p_{\mathcal{D}} = 2^{-24}$. The program was run in parallel on a cluster of 24 Pentium IV at 3 GHz. After 4 hours of computation, 544 distinguished points were computed and a useful collision occurred. About 9×10^9 steps were performed: this is a "lucky" run, since this is about $4.3\,p^{3/4}/m$. The memory requirement was about 1.5 MB on each node, mostly for the executable code, not the data. The amount of communication is reduced to a few KB between the nodes and a "master" node.

For that curve, $s_1/\sqrt{p} \approx -1.16$ and $s_2/p \approx 1.25$, so the curve was not exceptional, in the sense that it was not too close to the border of the area of Figure 1.

6 Curves of higher genus

The MCT algorithm can be extended to higher genus curves for which the characteristic polynomial of the Frobenius endomorphism is known modulo some integer m. In [11], different variants of this extension are studied and compared. Our extension of the original genus 2 MCT algorithm also applies to all these variants with a few modifications. Indeed, the number k of unknown coefficients in the characteristic polynomial can be larger than 2 and then a k-dimensional random walk should be designed in order to search for a collision in the intersection of two k-dimensional boxes. It is a tedious task to fill in the details, in particular the expected constants hidden in the $O(\ )$, but for fixed genus, we certainly lose only a constant factor compared to the memory-costly algorithms.

Another range of application of our method is the BSGS algorithm developed in [3] for counting points of the Jacobians of Picard curves. Without giving many details, let us mention that their algorithm ends by the search for a collision in two arithmetic progressions; therefore our algorithm applies almost directly to that case and should dramatically reduce the space complexity, which is the main drawback of their method.

7 Conclusion

We have presented a low-memory, parallelizable analogue of the algorithm by Matsuo, Chao and Tsujii for genus 2 hyperelliptic curves; the method works in other cases, such as the BSGS algorithm of [3] for Picard curves.

The main tool we used is a bidimensional pseudo-random walk. As usual with this kind of algorithms, it is impossible to make a rigorous analysis and heuristics and computer experiments are necessary to validate the approach. Our numerical data are positive in that sense, therefore our algorithm can be used when the memory constraint becomes problematic.

Acknowledgements

We thank Gérard Guillerm for giving us access to his PC cluster, Gilles Schaeffer for sharing his knowledge of the central limit theorem, and Edlyn Teske for many helpful comments on a first version of this paper. Part of the computations were done at LIX, on a cluster paid by the *ACI Cryptologie* funding.

References

1. MPICH: A portable implementation of MPI. `http://www-unix.mcs.anl.gov/mpi/mpich/`.
2. L. Adleman and M.-D. Huang. Counting points on curves and abelian varieties over finite fields. *J. Symbolic Comput.*, 32:171–189, 2001.
3. M. Bauer, E. Teske, and A. Weng. Point counting on Picard curves in large characteristic, 2003. Preprint.

4. I. Blake, G. Seroussi, and N. Smart. *Elliptic curves in cryptography*, volume 265 of *London Math. Soc. Lecture Note Ser.* Cambridge University Press, 1999.
5. W. Bosma and J. Cannon. *Handbook of Magma functions*, 1997. `http://www.maths.usyd.edu.au:8000/u/magma/`.
6. A. Bostan, P. Gaudry, and É. Schost. Linear recurrences with polynomial coefficients and computation of the Cartier-Manin operator on hyperelliptic curves. 2003. To appear in Proceedings Fq'7.
7. J.-P. Delescaille and J.-J. Quisquater. How easy is collision search? Application to DES. In J.-J. Quisquater, editor, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Comput. Sci.*, pages 429–434, 1990.
8. P. Gaudry and R. Harley. Counting points on hyperelliptic curves over finite fields. In W. Bosma, editor, *ANTS-IV*, volume 1838 of *Lecture Notes in Comput. Sci.*, pages 313–332. Springer–Verlag, 2000.
9. P. Gaudry and É. Schost. Construction of secure random curves of genus 2 over prime fields. In *Eurocrypt 2004*, To appear.
10. M.-D. Huang and D. Ierardi. Counting points on curves over finite fields. *J. Symbolic Comput.*, 25:1–21, 1998.
11. F. Izadi and K. Murty. Counting points on an abelian variety over a finite field. Preprint, 2003.
12. K. S. Kedlaya. Counting points on hyperelliptic curves using Monsky-Washnitzer cohomology. *J. Ramanujan Math. Soc.*, 16(4):323–338, 2001.
13. H. W. Lenstra, Jr., J. Pila, and C. Pomerance. A hyperelliptic smoothness test, II. *Proc. London Math. Soc.*, 84:105–146, 2002.
14. K. Matsuo, J. Chao, and S. Tsujii. An improved baby step giant step algorithm for point counting of hyperelliptic curves over finite fields. In C. Fiecker and D. Kohel, editors, *ANTS-V*, volume 2369 of *Lecture Notes in Comput. Sci.*, pages 461–474. Springer-Verlag, 2002.
15. J.-F. Mestre. Utilisation de l'AGM pour le calcul de $E(\mathbb{F}_{2^n})$. Letter to Gaudry and Harley, December 2000.
16. J. Pila. Frobenius maps of abelian varieties and finding roots of unity in finite fields. *Math. Comp.*, 55(192):745–763, October 1990.
17. J. M. Pollard. Monte Carlo methods for index computation mod p. *Math. Comp.*, 32(143):918–924, July 1978.
18. T. Satoh. The canonical lift of an ordinary elliptic curve over a finite field and its point counting. *J. Ramanujan Math. Soc.*, 15:247–270, 2000.
19. V. Shoup. *NTL: A library for doing number theory.* `http://www.shoup.net/ntl/`.
20. A. Stein and E. Teske. Explicit bounds and heuristics on class numbers in hyperelliptic function fields. *Math. Comp.*, 71:837–861, 2002.
21. A. Stein and E. Teske. The parallelized Pollard kangaroo method in real quadratic function fields. *Math. Comp.*, 71:793–814, 2002.
22. E. Teske. Speeding up Pollard's rho method for computing discrete logarithms. In J. P. Buhler, editor, *ANTS-III*, volume 1423 of *Lecture Notes in Comput. Sci.*, pages 541–554. Springer–Verlag, 1998.
23. E. Teske. Computing discrete logarithms with the parallelized kangaroo method. *Discrete Appl. Math.*, 130:61–82, 2003.
24. P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *J. of Cryptology*, 12:1–28, 1999.