ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ В ДИСКРЕТНОЙ МАТЕМАТИКЕ

УДК 512.772 DOI 10.17223/20710410/55/9

IMPLEMENTATION OF POINT-COUNTING ALGORITHMS ON GENUS 2 HYPERELLIPTIC CURVES BASED ON THE BIRTHDAY PARADOX[1]

N. Kolesnikov

Immanuel Kant Baltic Federal University, Kaliningrad, Russia

E-mail: NiKolesnikov1@kantiana.ru

Our main contribution is an efficient implementation of the Gaudry — Schost and Galbraith — Ruprai point-counting algorithms on Jacobians of hyperelliptic curves. Both of them are low memory variants of Matsuo — Chao — Tsujii (MCT) Baby-Step Giant-Step-like algorithm. We present an optimal memory restriction (a time-memory tradeoff) that minimizes the runtime of the algorithms. This tradeoff allows us to get closer in practical computations to theoretical bounds of expected runtime at $2.45\sqrt{N}$ and $2.38\sqrt{N}$ for the Gaudry — Schost and Galbraith — Ruprai algorithms, respectively. Here N is the size of the 2-dimensional searching space, which is as large as the Jacobian group order, divided by small modulus m, precomputed by using other techniques. Our implementation profits from the multithreaded regime and we provide some performance statistics of operation on different size inputs. This is the first open-source parallel implementation of 2-dimensional Galbraith — Ruprai algorithm.

Keywords: *hyperelliptic curve, Jacobian, point-counting, birthday paradox.*

РЕАЛИЗАЦИЯ АЛГОРИТМОВ ПОДСЧЁТА ТОЧЕК В ЯКОБИАНАХ ГИПЕРЭЛЛИПТИЧЕСКИХ КРИВЫХ РОДА 2, ОСНОВАННЫХ НА ПАРАДОКСЕ ДНЕЙ РОЖДЕНИЯ

Н. С. Колесников

Балтийский федеральный университет им. Иммануила Канта, г. Калининград, Россия

Представлена эффективная программная реализация алгоритма Годри — Шоста и его модификации Гэлбрайта — Рупрая для подсчёта точек в якобианах гиперэллиптических кривых. Эти алгоритмы представляют собой версии алгоритма Мацуо — Чао — Цуджия с малым использованием памяти и реализуют стратегию Гельфонда — Шенкса больших и малых шагов. Выводится оптимальный размер памяти, позволяющий минимизировать время работы указанных алгоритмов и получить на практике ожидаемое время их работы, близкое к теоретическим оценкам $2{,}45\sqrt{N}$ и $2{,}38\sqrt{N}$ для алгоритмов Годри — Шоста и Гэлбрайта — Рупрая соответственно. Здесь N — размер двумерной области поиска, равный порядку

[1] The publication was supported by the Russian Academic Excellence Project ‘5-100’ 2016–2020.

якобиана кривой, уменьшенному в m раз с помощью других методов. Предлагаемая реализация алгоритмов имеет многопоточный режим работы. Представлена статистическая зависимость времени работы от размера входных данных. Данная реализация алгоритма Гэлбрайта — Рупрая для размерности 2 является первой опубликованной многопоточной реализацией этого алгоритма с открытым исходным кодом.

Ключевые слова: *гиперэллиптическая кривая, якобиан, подсчёт точек, парадокс дней рождения.*

1. Introduction

Jacobians of hyperelliptic curves can be regarded as groups of large order that is difficult to compute. They have many applications in cryptography, such as DLP-based cryptosystems, where the efficiency of order computing algorithms is crucial. The MCT point-counting algorithm [1] is a natural attempt to adapt the general purpose BSGS algorithm to Jacobians of genus 2 hyperelliptic curves. This algorithm turned out to be unusable for cryptographic size instances due to its extremely high memory consumption. P. Gaudry and E. Schost have proposed [2] a low-memory version of the MCT algorithm and provided its experimental implementation. However, the memory limit in their implementation must be set by the user, and the authors do not give its optimal values.

Our main contribution is an efficient multithreaded implementation of the Gaudry — Schost point-counting algorithm with optimal memory settings that can be viewed as a time-memory tradeoff. We also implement an improvement of the Gaudry — Schost algorithm, proposed by S. Galbraith and R. Ruprai [3], and test the performance of both algorithms. This is the first open source implementation of the 2-dimensional Galbraith — Ruprai algorithm.

The paper is organized as follows. In Section 3, we give a review of the most efficient BSGS-like techniques proposed in [2, 3]. Both of them have heuristic complexity analyses based on the birthday paradox. These algorithms use the strategy of pseudo-random walks and store in RAM only a portion of points computed on “baby” and “giant” steps. In Section 4, we describe an efficient data structure to store distinguished points and an optimal storage size to reach the tradeoff between time and memory. Section 5 describes our implementation of the Gaudry — Schost and Galbraith — Ruprai algorithms and shows some statistics on their performance.

2. Preliminaries

We consider genus $g = 2$ hyperelliptic curves defined over a finite field $\mathbb{F}_q$ with prime $q > 2$. The curve is defined by the equation

$$\mathcal{C} : y^2 = x^{2g+1} + \sum_{i=0}^{2g} f_i x^i.$$

A set of all reduced divisors of a curve form a group $Jac(\mathcal{C})$, which is called a Jacobian of a curve. Its elements sometimes are regarded as “points”. The group law (point addition) on Jacobian is defined by Cantor’s addition formulas. We normally write reduced divisors in Mumford coordinates $D = (u(x), v(x))$, where $u(x)$ and $v(x)$ are polynomials, such that $\deg v(x) < \deg u(x) \leqslant g$. For further details on these algebraic structures, we refer to [4].

We denote by $\chi(T)$ the characteristic polynomial of the Frobenius endomorphism $\pi_{\mathcal{C}}$ on $Jac(\mathcal{C})$:

$$\chi(T) = T^4 - s_1T^3 + s_2T^2 - qs_1T + q^2.$$

3. Review of Birthday-paradox algorithms

3.1. Gaudry — Schost

Following P. Gaudry and E. Schost [5], before launching an exponential MCT point-counting algorithm, one can calculate $\#Jac(\mathcal{C}) \bmod m$, where $m = \ell_1^{k_1} \cdot \ldots \cdot \ell_s^{k_s}$, and ℓ_i are small primes. The cost of computing $\#Jac(\mathcal{C}) \bmod \ell$ takes $O(\log^7(\ell))$ operations [5, p. 2] in $\mathbb{F}_q$. In practice, we computed $\#Jac(\mathcal{C})$ modulo $\ell \leqslant 43$, which took ~ 40 Gb of RAM.

After gathering some modular information about $\#Jac(\mathcal{C})$, we proceed to the exponential Gaudry — Schost point-counting algorithm. The desired order of the Jacobian can be computed by substituting $T = 1$ into Frobenius characteristic polynomial

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

Thus, we have to find two values s_1 and s_2. We write them in the form $s_i = \overline{s_i} + m\widetilde{s_i}$, $i = 1, 2$, assuming that we have $\overline{s_1}, \overline{s_2}$ precomputed. This reduces our search space for a tuple of unknowns $(\widetilde{s_1}, \widetilde{s_2})$ to the following bounds [2, p. 3]:

$$-\frac{\sqrt{q}}{m} \leqslant \widetilde{s_1} \leqslant \frac{\sqrt{q}}{m}, \quad -\frac{2q}{m} \leqslant \widetilde{s_2} \leqslant \frac{6q}{m}. \tag{2}$$

Then we proceed as follows. We choose a random divisor $D \in Jac(\mathcal{C})$ and try to compute its order. As the order $\operatorname{ord}(D)$ divides the group order $\#Jac(\mathcal{C})$, we have $\chi(1)D = 0$. Combining this equation with (1) gives

$$(q^2 + 1 - \overline{s_1}(q+1) + \overline{s_2})\, D + (-\widetilde{s_1}(q+1) + \widetilde{s_2})\, m \cdot D = 0. \tag{3}$$

We form the two sets of divisors: W (“Wild”) and T (“Tame”), and enumerate elements in these sets until we find a point belonging to both W and T (a collision). A constant K' contains known terms of (3) and a correction term to make the bounds (2) on $\widetilde{s_1}, \widetilde{s_2}$ symmetric. We later subtract this correction term once we find a collision:

$$\begin{gathered}
W = \{K' \cdot D + (-\sigma_1(q+1) + \sigma_2)\, m \cdot D : (\sigma_1, \sigma_2) \in R\}, \\
T = \{(-\sigma_1(q+1) + \sigma_2)\, m \cdot D : (\sigma_1, \sigma_2) \in R\}, \\
R = [B_{1,\min}, B_{1,\max}] \times [B_{2,\min}, B_{2,\max}], \\
B_{1,\min} = -\frac{\sqrt{q}}{m}, \quad B_{1,\max} = \frac{\sqrt{q}}{m}, \quad B_{2,\min} = -\frac{2q}{m}, \quad B_{2,\max} = \frac{6q}{m}, \\
K' = q^2 + 1 - \overline{s_1}(q+1) + \overline{s_2} + m\left(-\left\lfloor \frac{B_{1,\min} + B_{1,\max}}{2} \right\rfloor (q+1) + \left\lfloor \frac{B_{2,\min} + B_{2,\max}}{2} \right\rfloor\right).
\end{gathered}$$

Two colliding points $D_W = (\sigma_{1W}, \sigma_{2W})$ and $D_T = (\sigma_{1T}, \sigma_{2T})$ give us the unknown values $\widetilde{s_1}$, $\widetilde{s_2}$ by the following equations derived from (3):

$$\widetilde{s_i} = \sigma_{i,W} - \sigma_{i,T} + \left\lfloor \frac{B_{i,\min} + B_{i,\max}}{2} \right\rfloor, \quad i = 1, 2.$$

The time complexity of the algorithm depends on the cardinality of the intersection $|W \cap T|$ which lies in the interval $[0.25|R|..|R|]$ depending on the curve given. The expected number of points to be constructed until we get a collision is $\sqrt{\pi \cdot |W \cap T|}$, that follows from the Theorem 1 below. Thus, the expected number of points in the best, worst and average [6] cases are $1.77|R|$, $3.54|R|$ and $2.43|R|$ respectively.

Theorem 1 (Tame-Wild birthday paradox). Suppose that we have two urns both containing M balls numbered from 1 to M. The first urn contains only white balls, the second urn contains only red balls. We are choosing the ball uniformly at random in course from the first and the second urn, save its number and color, and return the ball to the urn. Then the expected number of selections until we get two colliding numbers of different colors is

$$\sqrt{\pi M} + O(1).$$

We denote by $\mathbb{P}(M, m_1, m_2)$ the probability that after m_1 steps in urn 1 and m_2 steps in urn 2 no matches were found. K. Nishimura and M. Sibuya prove in [7] that if we are restricted to $m_1 = m_2 = m = O(\sqrt{M})$, $M \to \infty$, this probability tends to

$$\mathbb{P}(M, m, m) \to \exp\left(-\frac{m^2}{M}\left[1 + O\left(\frac{1}{\sqrt{M}}\right)\right]\right) \approx \exp\left(-\frac{m^2}{M}\right).$$

Let X be a random variable that represents the number of selections of any urn before we get a collision. Then the cumulative distribution function is

$$F_X(m) = 1 - \mathsf{P}[X > m] = 1 - \mathbb{P}(M, m, m).$$

This fact is used to calculate the expectation of X that gives

$$\mathbb{E}(X) = \sqrt{\pi M}/2.$$

.

Remark 1. One should note that if we fix the input parameters $(\mathbb{F}_q, \mathcal{C}, m)$ and run the Gaudry — Schost point-counting algorithm several times, the best and the worst running time estimates could not be compared with the values $1.77|R|$ and $3.54|R|$. To estimate the deviation on running time with the fixed input, we need to compute the variance $\mathrm{Var}(X)$ of the random variable above. This deviation we will use in practical experiments:

$$\begin{aligned}\mathbb{E}(X^2) &= \sum_{i=1}^{\infty} i^2 \cdot \mathsf{P}[X = i] = \sum_{i=1}^{\infty} i^2 \cdot \mathsf{P}[X > i-1] - \sum_{i=1}^{\infty} i^2 \cdot \mathsf{P}[X > i] = \\ &= \sum_{i=0}^{\infty} (i+1)^2 \cdot \mathsf{P}[X > i] - \sum_{i=1}^{\infty} i^2 \cdot \mathsf{P}[X > i] = \\ &= \sum_{i=0}^{\infty} (2i+1)\, \mathsf{P}[X > i] = \int_{i=0}^{\infty} (2i+1) \exp\left(-i^2/M\right) di = \\ &= \int_{x=0}^{\infty} 2x \cdot \exp\left(-x^2/M\right) dx + \int_{x=0}^{\infty} \exp\left(-x^2/M\right) dx = M + \frac{\sqrt{\pi M}}{2},\end{aligned}$$

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = M \cdot \left(1 - \frac{\pi}{4}\right) + \frac{\sqrt{\pi M}}{2}.$$

Thus, the standard deviation for X is given by

$$\sqrt{\mathrm{Var}(X)} \underset{M \to \infty}{=} \sqrt{1 - \pi/4} \cdot \sqrt{M} \approx 0.46\sqrt{M}.$$

Gaudry and Schost in [5] also present an approach of random walks and distinguished points that significantly reduces the memory requirement of the algorithm. The idea is to name a portion of points in the search space R as “distinguished” points. This is done by selecting an appropriate hash function and looking at some bits of each hash value.

As before, we choose a random divisor $D \in Jac(\mathcal{C})$, calculate its hash value $h(D)$. This value determines the direction of a random walk. The next divisor we choose is $D + \mathcal{O}_{h(D)}$, where "$\mathcal{O}$" is a short list of precomputed shifts that defines the behavior of a random walk. We continue the walk unless a distinguished point is hit. As soon as this happens, the distinguished point is saved to an appropriate W or T list. Although the points in the constructed chain are not taken from the search space uniformly at random, the complexity analysis, based on the Tame-Wild birthday paradox, heuristically remains valid. However, an average runtime and memory requirements vary depending on the random walk parameters. We discuss the optimal random walk set up in Section 4.

3.2. Galbraith — Ruprai

S. Galbraith and R. Ruprai proposed in [3] an improvement for the Gaudry — Schost algorithm, that reduces the search space R in a tricky way, and makes the cardinality of intersection $|W \cap T|$ constant for all curve instances. As a result, the expected number of points to be constructed in any of the cases: best, worst, or average is invariant and equals to $2.38|R|$. The notions of random walks, distinguished points, and its complexity analysis remain the same. The search space and Tame-Wild sets are defined as follows: the Tame searching rectangle is reduced in length and width by 2/3 times, while the Wild searching rectangle is constructed as a union of 4 disjointed "corners" of R:

$$\begin{gathered}
R_T = [2/3B_{1,\min}, 2/3B_{1,\max}] \times [2/3B_{2,\min}, 2/3B_{2,\max}],\\
R_W = [B_{1,\min}, B_{1,\min} + (B_{1,\max} - B_{1,\min})/3] \times [B_{2,\max}, B_{2,\max} - (B_{2,\max} - B_{2,\min})/3] \cup\\
\cup\, [B_{1,\max}, B_{1,\max} - (B_{1,\max} - B_{1,\min})/3] \times [B_{2,\max}, B_{2,\max} - (B_{2,\max} - B_{2,\min})/3] \cup\\
\cup\, [B_{1,\min}, B_{1,\min} + (B_{1,\max} - B_{1,\min})/3] \times [B_{2,\min}, B_{2,\min} + (B_{2,\max} - B_{2,\min})/3] \cup\\
\cup\, [B_{1,\max}, B_{1,\max} - (B_{1,\max} - B_{1,\min})/3] \times [B_{2,\min}, B_{2,\min} + (B_{2,\max} - B_{2,\min})/3],\\
W' = \{K' \cdot D + (-\sigma_1(q+1) + \sigma_2)\, m \cdot D : (\sigma_1, \sigma_2) \in R_W\},\\
T' = \{(-\sigma_1(q+1) + \sigma_2)\, m \cdot D : (\sigma_1, \sigma_2) \in R_T\}.
\end{gathered}$$

Random walks on T' and W' operate similarly to the Gaudry — Shost algorithm. The only difference is that we choose a random divisor that initiates a chain in one of the corners of R_W and do not jump to another corner. The number of distinguished points belonging to any corner of R_W is 1/4 of distinguished points belonging to T. We also change the step size of precomputed shifts "O" to prohibit overjumping the searching area, this problem is the point of discussion in Section 4.

4. Time-memory tradeoff for Gaudry — Schost algorithm

First, we describe the aspects of our implementation. We will use the following notations:

— E is the expected number of distinguished points to be stored, i.e., the expected memory requirement of the algorithm. Note that the actual memory size to store one distinguished point is about $7 \log q + O(1)$ bits. The associated data structure contains 5 long integers of size q, representing a divisor in Mumford coordinates $(u_0, u_1, u_2, v_0, v_1)$, 2 long integers $\sigma_1, \sigma_2 \in R$, encoding a position of the point in a searching rectangle, a hash value of a divisor, that is a 32-bit integer, and a Boolean value indicating Wild or Tame walk;

— θ is the probability for a random point $D \in \{T, W\}$ to be a distinguished point. It is easy to see that the expected length of a chain of a random walk is $U = 1/\theta$. Following Gaudry, we use a 32-bit hash function, that has a weak correlation with the arithmetic properties of a point. We consider a point to be distinguished iff some bits in a hash value are equal to zero. Thus, the probability of being distinguished can be customized stepwise starting from 1/32, 1/16, 3/32, and so on.

We store all distinguished points in a single array. We do not sort this array directly because the points themselves are rather "heavy" as noted above, and their relocation will lead to suboptimal time. Instead of this, we store an additional vector of pointers. This vector addresses the elements sorting them by hash value. That is why the hash value is stored together with a point itself. As soon as a distinguished point is hit, we save it to the end of the main array, that has $O(1)$ time complexity. Then we find an appropriate position for this point in a sorted list of pointers, which is done in $O(\log E)$ by binary search. Then we insert a new pointer to the list, which also has time complexity $O(1)$.

Proposition 1. The time complexity of our Gaudry — Schost implementation (in average case for a random curve) is given by

$$T = \alpha + \frac{1}{\theta} + E \cdot \log_2 E \text{ (operations in } \mathbb{F}_q), \tag{4}$$

where $\alpha = 2.43(1+\epsilon)\sqrt{|R|}$ for Gaudry — Schost algorithm; $\alpha = 2.38(1+\epsilon)\sqrt{|R_T|}$ for its Galbraith — Ruprai improvement.

Proof. The value α in (4) is an expected number of points in the search space to be enumerated until we find a collision. Evaluation of α relies on the Tame-Wild birthday paradox and can be found in [2, 6]. To start a new random walk, we choose a point uniformly at random, that takes constant time. Each step in a random walk requires one group operation in $Jac(\mathcal{C})$, which can be done in time $O(1)$ of operations in $\mathbb{F}_q$ by applying explicit formulas [8]. There are two kinds of "bad" points that could not be accounted when applying the Tame-Wild birthday paradox:

1) points that give a cycle in a random walk. Once we get a loop, a random walk will never hit a distinguished point and must be aborted. P. C. van Oorschot and M. J. Wiener have shown [9] that if we restrict the maximum length of a chain to $20/\theta$, then the number of such "bad" points is at most $5 \cdot 10^{-8}$;
2) points that lie outside the search space R, R_T or R_W. To reduce the number of such overjumps, we follow Gaudry — Schost [2] and make the precomputed shifts "O" not greater than
$$\ell_2 = \frac{(B_{2,\max} - B_{2,\min})\,\theta}{10}, \quad \ell_1 = \frac{(B_{1,\max} - B_{1,\min})\sqrt{\theta}}{9}$$
for both directions. Thus, the expected length of a chain is one tenth of the search space R in both σ_1 and σ_2 directions. "Bad" points of this type give a correction factor $(1+\epsilon)$ to the Tame-Wild birthday paradox theorem, where ϵ, following [3], is a small factor between 0.02 and 0.04.

The term $1/\theta$ in (4) takes into account the length of the last chain, because a collision may occur at any intermediate point of a walk. However, the walk continues up to the distinguished point. The last term $E \cdot \log_2 E$ is the time wasted on binary search in W or T lists to find a collision. ■

Proposition 2. The memory restriction that gives us a time-memory tradeoff for our implementation is

$$E = \sqrt{\frac{2\alpha \ln 2}{\log(2\alpha \ln 2)}},$$

where α is defined in Proposition 1.

Proof. Rewrite equation (4), assuming $1/\theta$ is the average length of a chain and E is the number of chains constructed:

$$T = T(E) = \alpha + \frac{\alpha}{E} + E \cdot \log_2 E.$$

Find the minimum value of time function $T(E)$:

$$T'(E) = -\frac{\alpha}{E^2} + \log_2 E + \frac{1}{\ln 2} = 0.$$

The only critical point that is a point of local minimum for a function $T(E)$ is

$$E = \frac{\sqrt{D}}{\sqrt{W(D)}} = \sqrt{e^{W(D)}} \approx \sqrt{e^{\log D - \log\log D}} = \sqrt{\frac{D}{\log D}},$$

where $D = 2\alpha \ln 2$ and W is a Lambert W function. ■

5. Implementation and tests

5.1. General description

We present an optimized implementation of Gaudry — Schost and Galbraith — Ruprai point-counting algorithms. This is a fork from Gaudry's NTLJac2 [10] package. This package is implemented in C++ and extends the Number Theory Library (NTL) with special tools for Jacobians of genus 2 hyperelliptic curves. It contains data structures to represent divisors on a curve and its arithmetic. On top of Gaudry's package, we added the efficient data storage for the distinguished points as described in Section 4 and implemented an improvement proposed by Galbraith and Ruprai. Moreover, we made our implementation multithreaded.

5.2. Runs on different curves

First, we collected some statistics to test Gaudry — Schost time complexity and compare it with those of Galbraith — Ruprai improvement. We fixed a field $\mathbb{F}_q$ with $q = 2^{45} + 59$, randomly generated $N = 300$ curves and for each of them computed s_1, s_2 modulo $m = 3 \cdot 5 \cdot 7 \cdot 11 = 1155$. As noted in Remark 1, it is not correct to run an algorithm once to evaluate its runtime. So for each curve, we did $n = 30$ runs and calculated an average runtime and number of stored distinguished points. Remark 1 shows that for $n = 30$ experiments the standard deviation on the number of distinguished points reduces from $0.92\sqrt{M}$ to $\frac{0.92}{\sqrt{n}}\sqrt{M} \approx 0.17\sqrt{M}$. Thus, the number of distinguished points constructed for each curve deviates on average $0.17/2.54 \approx 6.7\,\%$ of the theoretical value. This deviation is still quite significant but allows to select "the best" and "the worst" curve instances.

We did the same experiment for the Gaudry — Schost algorithm and its improvement. All of the $N = 300$ curves are sorted by the quantity of distinguished points constructed on average. The number of a curve tested is placed on the X-axis, whereas the quantity of stored points (Fig. 1, a) or overall elapsed time (Fig. 1, b) is on Y-axis. The expected quantity of distinguished points to be stored is $E = 2282$ for the above input, which agrees with our statistics.

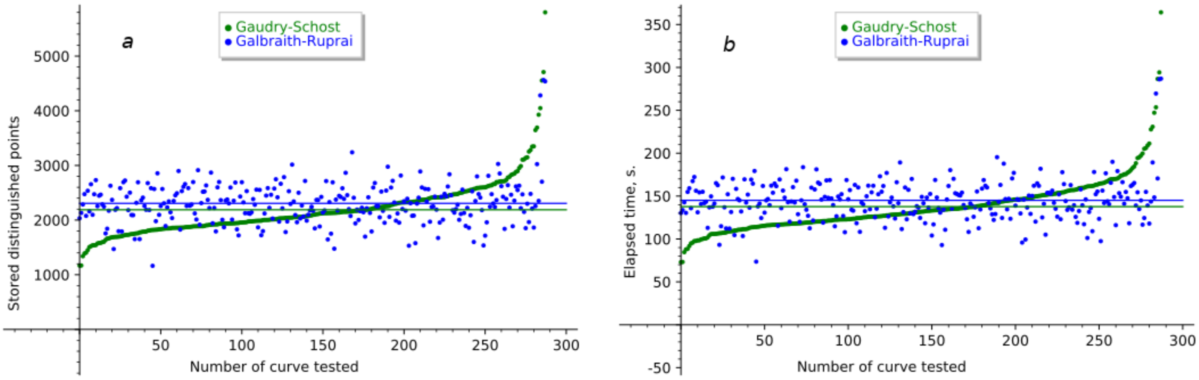

Fig. 1. Gaudry — Schost and Galbraith — Ruprai algorithms. Performance on different curves: a — memory; b — time

5.3. Several runs on the same curve

The aim of running our software on a fixed curve is to test the time-memory tradeoff bound, stated in Proposition 1. We selected $N = 3$ curves from the test above, that are close to the best, worst, and average cases of the Gaudry — Schost algorithm. We modified the parameter E to $E/2$, $E/4$, $2E$, $4E$ and $E = 1000$ as proposed by Gaudry. For each value, we run the software $n = 100$ times. This guarantees the deviation of the experimental number of selected points from the theoretical value by 3.6 % on average. According to proposition 2, the time-memory tradeoff for our input data requires to store $E = 2282$ points for Gaudry — Schost algorithm and $E = 1890$ for its Galbriath — Ruprai improvement (Tables 1 and 2).

All performance tests described above have been executed on Xeon E-2146G 6C 3.50GHz, system RAM available is 16GB. The software is compiled with gcc 9.3.0 compilers under Ubuntu 20.04 operating system.

Table 1

Gaudry — Schost performance on different memory restrictions

		$E = 456$	$E = 1141$	$E = 2282$ (tradeoff)	$E = 4564$	$E = 11410$	$E = 1000$
Curve 1	Dist. points	494	1037	1892	3828	7553	1086
	Time, s.	104.8	109.6	98.9	100.3	98.8	112.8
Curve 120	Dist. points	493	1195	1907	4183	8006	1207
	Time, s.	103.5	124.8	99.1	109.4	106.3	126.7
Curve 300	Dist. points	623	1180	2248	4646	10986	1326
	Time, s.	131.9	123.5	117.2	123.4	144.6	138.6

Table 2

Galbraith — Ruprai improvement on different memory restrictions

		$E = 378$	$E = 945$	$E = 1890$ (tradeoff)	$E = 3780$	$E = 9450$
Curve 1	Dist. points	491	1121	1910	4572	20354
	Time, s.	102.8	117.3	100.3	120.7	136.9
Curve 120	Dist. points	660	1239	2082	4237	19855
	Time, s.	138.0	129.7	109.6	108.8	131.5
Curve 300	Dist. points	635	1140	2217	4113	19233
	Time, s.	133.9	119.9	116.2	112.4	127.6

6. Conclusion

We presented efficient implementations of two BSGS-like point-counting algorithms based on the birthday paradox. A time-memory tradeoff has been provided for both algorithms. It allows us to minimize the runtime by allocating enough memory. We did not test our implementation on cryptographic size input, as we were unable to precompute $\overline{s_1}, \overline{s_2}$ for sufficiently large moduli m. However, we believe our implementation might be useful for computations with any size of Jacobians, in combination with other techniques. The source code of our implementation can be found here: `https://github.com/kn02262/Jac2pc`.

REFERENCES

1. *Matsuo K., Chao J., and Tsujii S.* An improved baby step giant step algorithm for point counting of hyperelliptic curves over finite fields. LNCS, 2002, vol. 2369, pp. 461–474.
2. *Gaudry P. and Schost E.* A low-memory parallel version of Matsuo, Chao and Tsujii's algorithm. LNCS, 2004, vol. 3076, pp. 208–222.
3. *Galbraith S. and Ruprai R.* An improvement to the Gaudry — Schost algorithm for multidimensional discrete logarithm problems. LNCS, 2009, vol. 5921, pp. 368–382.
4. *Cohen H., Frey G., Avanzi R., et al.* Handbook of Elliptic and Hyperelliptic Curve Cryptography. CRC Press, 2005.
5. *Gaudry P. and Schost E.* Genus 2 point counting over prime fields. J. Symbolic Comput., 2012, vol. 47, iss. 4, pp. 368–400.
6. *Ruprai R. S.* Improvements to the Gaudry — Schost Algorithm for Multidimensional Discrete Logarithm Problems and Applications. PhD Thesis, Department of Mathematics, Royal Holloway University of London, 2010. `https://www.math.auckland.ac.nz/~sgal018/Ruprai-thesis.pdf`.
7. *Nishimura K. and Sibuya M.* Probability to meet in the middle. J. Cryptology, 1990, no. 2, pp. 13–22.
8. *Hisil H. and Costello C.* Jacobian coordinates on genus 2 curves. J. Cryptology, 2017, vol. 30, iss. 2, pp. 572–600. `https://doi.org/10.1007/s00145-016-9227-7`.
9. *Van Oorschot P. C. and Wiener M. J.* Parallel collision search with cryptanalytic applications. J. Cryptology, 2013, vol. 12, pp. 1–28.
10. *Gaudry P.* C++ NTLJac2 Library, 2003, `http://www.lix.polytechnique.fr/Labo/Pierrick.Gaudry/NTLJac2`.