Improvements to Jacobian Arithmetic in Global Function Fields

Vincent Macri(✉), Michael Jacobson Jr., and Renate Scheidler

University of Calgary, Calgary, Canada
{vincent.macri,jacobs,rscheidl}@ucalgary.ca

Abstract. We present two improvements to arithmetic in the Jacobian of global function fields based on the approach of Hess. The first reduces the number of expensive reduction steps by optimizing for typical inputs rather than worst-case behavior, assuming the function field contains a degree-one place. This results in an expected asymptotic speed-up by a factor that is logarithmic in the genus. The second introduces a memory–time trade-off that speeds up computations by caching frequently used intermediate results. Our asymptotic analysis and empirical experiments show that our improved algorithms are significantly faster in practice than previously published methods. To the best of our knowledge, our publicly-available software implementation of Jacobian arithmetic is the first to support unique representatives of divisor classes.

Keywords: Global function field · Algebraic curve · Divisor · Jacobian

1 Introduction

1.1 Motivation

Elliptic curves are the subject of numerous notoriously challenging open problems, including the Koblitz–Zywina conjecture, the Lang–Trotter conjecture, the Sato–Tate conjecture, and perhaps most famously, the Birch and Swinnerton-Dyer conjecture [19,23]. While these conjectures are about elliptic curves over $\mathbb{Q}$, generating numerical evidence for (or against) them requires considering a given curve over finite fields $\mathbb{F}_q$ for many values of q and computing the number of points on the curve. Efficient point arithmetic is essential in this endeavour. This motivates the desire to compute the size of the group of points on an elliptic curve over a finite field. Although proofs of these conjectures are out of reach, there is much computational evidence supporting them.

A natural next step is to consider generalizations of these conjectures to curves that are not necessarily elliptic, and see whether computational evidence supports or refutes the generalized conjectures. In the more general setting, the points on the curve no longer form a group: instead, elements of the Jacobian are equivalence classes of degree zero divisors. Hence, rather than counting points over many finite fields, one instead needs to compute the order of the Jacobian of the curve over all these fields. In order to tackle this and similar problems,

L. Batina and F. Özbudak (Eds.): WAIFI 2026, LNCS 16611, pp. 111–128, 2026.
https://doi.org/10.1007/978-3-032-27574-5_8

it is essential to have fast algorithms for computing in the Jacobian group of a curve or its associated function field. Moreover, for algorithmic efficiency, it is beneficial if this arithmetic should be performed in a way that gives unique representatives of elements of the Jacobian group.

The most recent efficient algorithm for computing in the Jacobian with unique representatives of degree divisor classes is the approach of maximally reduced divisors from [7]. Other recent approaches to Jacobian arithmetic such as [10–12] do not give unique representatives of Jacobian elements.

1.2 Our Contributions

In [7], Hess showed how to compute in the divisor class group, which contains the Jacobian as a subgroup. By specializing to the Jacobian and requiring the existence of a degree one place (a restriction which is nearly always satisfied), we develop an improved algorithm that supports unique representation of divisor classes and is significantly faster than the approach recommended in [7]. We also demonstrate how effective use of caching can speed up Jacobian arithmetic even further, especially if the function field contains a degree one infinite place. Our improved Jacobian algorithm is both faster asymptotically for typical inputs, and faster in practice when implemented. Complexity analysis indicates that our algorithm should give a speedup of approximately $\log_2(g)/2$, where g is the genus of the function field. A performance comparison confirms this: for example, for degree 3 function fields of genus 100, our implementation of our improved algorithm is approximately three times faster than our implementation of the unique Jacobian arithmetic algorithm suggested in [7]. An implementation of our algorithm is available in SageMath [22], since the 10.9.beta7 release.

1.3 Outline of the Paper

In Sect. 2 we establish notation and summarize the necessary background material. Section 3 describes our modifications to Hess' maximally reduced divisors from [7] that facilitate our improved algorithms for Jacobian arithmetic presented in Sect. 4. We conduct a theoretical complexity analysis in Sect. 5 before analyzing the actual performance of our implementations in Sect. 6 and proposing directions for future research on Sect. 7.

2 Background

For details, the reader is referred to [18]. Throughout, let K be a finite field and F/K a geometric global function field of genus g. We write $F = K(x, y)$ with $x, y \in F$, where $K(x)$ is a rational function field and y has minimal polynomial $f(t) = t^n + \sum_{i=0}^{n-1} t^i a_i(x) \in K[x, t]$, with $a_i(x) \in K[x]$ for $0 \leq i \leq n-1$. Then $F/K(x)$ has degree n in t and $f(y) = 0$. The quantity

$$C_f := \max \left\{ \left\lceil \frac{\deg(a_i)}{n-i} \right\rceil : 0 \leq i \leq n-1 \right\} \tag{1}$$

determines the size of F and will appear in our complexity statements [7, p. 427].

We denote the set of places of F by $\mathbb{P}(F)$, partitioned into the infinite places (the poles of x) and the finite places of F. The finite and infinite maximal orders of $F/K(x)$ are denoted by $\mathfrak{O}_{F,0}$ and $\mathfrak{O}_{F,\infty}$ respectively. Divisors of F are formal finite sums $D = \sum_{P\in\mathbb{P}(F)} v_P(D)P$, where $v_P(D) \in \mathbb{Z}$ is the valuation of D at P. We write $\mathrm{supp}(D) = \{P \in \mathbb{P}(F) : v_P(D) \neq 0\}$ (the support of D) and let $\deg(D)$ denote the degree of D. The respective sub-sums of D over the finite places and the infinite places in $\mathrm{supp}(D)$ are denoted by D^0 and D^∞, so $D = D^0 + D^\infty$.

The principal divisor of a non-zero function $a \in F$ is denoted $\mathrm{div}(a)$. Two divisors $D, D' \in \mathrm{Div}(F)$ are (linearly) equivalent, denoted $D \equiv D'$, if they differ by a principal divisor. Let $\mathrm{Div}(F)$ and $\mathrm{Div}^0(F)$ denote the groups of divisors and degree zero divisors of F, respectively, and let $\mathrm{Cl}(F)$ and $\mathrm{Cl}^0(F)$ denote the groups of divisor classes and degree zero divisor classes of F, respectively, under linear equivalence. The group $\mathrm{Cl}^0(F)$ is also referred to as the *Jacobian* of F and is the main protagonist of our work herein.

The Riemann-Roch space of a divisor $D \in \mathrm{Div}(F)$ is the finite-dimensional K-vector space $\mathcal{L}(D) = \{a \in F : D + \mathrm{div}(a) \geq 0\} \cup \{0\}$; its K-dimension is denoted by $\ell(D)$. A partial order on $\mathrm{Div}(F)$ is given by $D \leq D'$ if $v_P(D) \leq v_P(D')$ for all $P \in \mathbb{P}(F)$. Riemann-Roch spaces and their dimensions are monotonic with respect to this partial order; this monotonicity will play a key role in our main algorithm (Algorithm 1).

Lemma 1. *[18, Lemma 1.4.8] Let $D, D' \in \mathrm{Div}(F)$ with $D \leq D'$. Then $\mathcal{L}(D) \subseteq \mathcal{L}(D')$ and $\ell(D') - \ell(D) = \dim(\mathcal{L}(D')/\mathcal{L}(D)) \leq \deg(D') - \deg(D)$.*

We conclude this section with the notion of a maximally reduced divisor as presented in [7]. For convenience, we restate the definition here.

Definition 1. *[7, Definition 8.1] Let $A \in \mathrm{Div}(F)$ with $\deg(A) \geq 1$. A divisor $\tilde{D}$ is called* maximally reduced along A *if $\tilde{D} \geq 0$ and $\ell(\tilde{D} - sA) = 0$ for all integers $s \geq 1$. The representation of a divisor D as $D = \tilde{D} - rA - \mathrm{div}(a)$, with $\tilde{D}$ maximally reduced along A, $r \in \mathbb{Z}$, and $a \in F^*$ is called a* maximal reduction of D along A.

Hess [7, Proposition 8.2] showed that if $\deg(A) = 1$ then the maximal reduction of a divisor D along A is unique.

3 Unique Hess Representation

The notion of maximally reduced divisors (Definition 1) was originally defined for elements of $\mathrm{Cl}(F)$, but restricting to $\mathrm{Cl}^0(F)$ and imposing the restriction that A is a degree one place allows for us to achieve improved algorithmic performance. We note that the following notion is restricted to degree zero divisors.

Definition 2. *Let $D \in \mathrm{Div}^0(F)$ and $A \in \mathbb{P}(F)$ with $\deg(A) = 1$. The* Hess-reduction of D (along A) *is the maximal reduction of D along A. We call D* Hess-reduced (along A) *if D is equal to its Hess-reduction (along A) and refer to it as the* Unique Hess representation *of its divisor class.*

By [7, Proposition 8.2], the Hess-reduction of a degree zero divisor is unique. Here, A can be finite or infinite, but the latter case will allow for more efficient implementations later on as it creates additional opportunities to exploit caching of intermediate results. In practice, requiring the existence of a degree one place is a very mild restriction. By the Hasse-Weil Bound [18, Theorem 5.2.3], every function field $F/\mathbb{F}_q$ of genus g has a degree one place if q is sufficiently large relative to g. Hence, the existence of a degree one place can be guaranteed by extending scalars to a sufficiently large extension of $\mathbb{F}_q$. Even for prime fields $\mathbb{F}_q$, the probability that a function field has no degree one place is quite low; see [8] for a heuristic argument.

We require two key results for Hess-reduced divisors.

Proposition 1. *Let $A \in \mathbb{P}(F)$ with $\deg(A) = 1$, and let $\tilde{D} \in \mathrm{Div}^0(F)$ be Hess-reduced along A. Then $A \notin \mathrm{supp}(\tilde{D})$.*

Proof. Since $\tilde{D}$ is maximally reduced along A, we have $\ell(\tilde{D} - A) = 0$. It follows that $\tilde{D} - A \not\geq 0$, so $v_A(\tilde{D}) \leq 0$. On the other hand $\tilde{D} \geq 0$, so $v_A(\tilde{D}) = 0$. □

In [7], Hess noted that for a divisor $\tilde{D}$ maximally reduced along A, we have $\ell(\tilde{D}) \leq \deg(A)$ and $\deg(\tilde{D}) < g + \deg(A)$. We strengthen these statements for the setting of Hess-reduced divisors.

Proposition 2. *Let $A \in \mathbb{P}(F)$ with $\deg(A) = 1$, and let $D = \tilde{D} - rA$ be Hess-reduced along A. Then $\deg(\tilde{D}) = r$ and $0 \leq r \leq g$.*

Proof. We have $\deg(\tilde{D}) = \deg(rA) = r$. Since $\tilde{D}$ is maximally reduced along A, we have $\tilde{D} \geq 0$ and $\ell(\tilde{D} - A) = 0$, so Lemma 1 yields

$$\ell(\tilde{D}) = \ell(\tilde{D}) - \ell(\tilde{D} - A) \leq \deg(\tilde{D}) - \deg(\tilde{D} - A) = \deg(A) = 1.$$

Riemann's Theorem [18, Theorem 1.4.17] implies $\deg(\tilde{D}) \leq \ell(\tilde{D}) + g - 1 \leq g$. □

4 Improvements to the Hess Reduction Algorithm

In this section, we introduce two new modifications to the algorithm of [7] for computing maximal reductions that effect substantial efficiency improvements when restricting to degree zero divisors. The first employs a search strategy for finding maximal reductions that is different from Hess's and performs better in practice. The second improvement introduces caching of intermediate computational results which effects a further speed-up.

4.1 Improvements to Reduction Algorithm

In [7], Hess provided an efficient algorithm for computing a basis of a Riemann-Roch space and described how to use it for finding maximal reductions of divisors. After specializing to degree zero divisors and the framework of Unique Hess representations in the Jacobian of F, this task translates to the following problem (see [14, Section 4.1] for details).

Problem 1 (`HR-Min`). Let $A \in \mathbb{P}(F)$ with $\deg(A) = 1$. Given $D \in \mathrm{Div}^0(F)$, find the minimal integer $0 \leq r \leq g$ such that $\ell(D + rA) = 1$.

By the monotonicity of ℓ, an equivalent problem is to find the maximal integer r' such that $\ell(D + r'A) = 0$. Then $r = r' + 1$ solves `HR-Min`.

Once a solution to `HR-Min` for an input divisor $D \in \mathrm{Div}^0(F)$ is found, it is fairly straightforward to compute the Hess-reduction of D, assuming we can compute Riemann-Roch bases. Suppose that r solves `HR-Min` for D and that we can compute a non-zero element $a \in \mathcal{L}(D + rA)$, which is then a basis of this space. Then $\mathrm{div}(a)$ is unique because $\ell(D+rA) = 1$. Moreover, $D+rA+\mathrm{div}(a) \geq 0$, so setting $\tilde{D} \coloneqq D + rA + \mathrm{div}(a)$, we see that $\tilde{D} \geq 0$ and $\ell(\tilde{D} - sA) = 0$ for all integers $s \geq 1$. So $\tilde{D}$ is maximally reduced along A, and hence $D \equiv \tilde{D} - rA$ is the unique Hess-reduction of D along A. See Algorithm 2 for an algorithmic description of this process.

Finding the solution r to `HR-Min` can be accomplished by trying each possible candidate value for r between 0 and g, but each trial requires computing a basis of a Riemann-Roch space, which is by far the most expensive part of the computation. In [7], Hess suggested to find r via binary search, which takes $O(\log g)$ iterations, but Hess was working in $\mathrm{Cl}(F)$ rather than $\mathrm{Cl}^0(F)$. When working over $\mathrm{Cl}^0(F)$, a heuristic argument [14, Subsection 4.2.1] shows that with high likelihood, for a randomly chosen $D \in \mathrm{Div}^0(F)$, the solution to `HR-Min` is $r = g$. Empirically, we find that for $K = \mathbb{F}_q$, the value $r = g$ solves `HR-Min` with probability approaching $1 - q^{-1}$ as q grows. This stems from the observation that a random divisor of degree at most g is expected to have degree exactly g with the same likelihood that a random polynomial of degree at most g in $\mathbb{F}_q[x]$ is expected to have degree exactly g, which is just the probability that its coefficient at x^g is non-zero. Furthermore, the same heuristic suggests that when $r = g$ is not the solution to `HR-Min`, the next most likely candidate is $r = g - 1$, then $g - 2$, and so on, with the solution $r = 0$ corresponding to the case when D is principal.

This observation suggests that a linear search in decreasing order to solving `HR-Min` should outperform the binary search approach suggested in [7], by significantly decreasing the number of Riemann-Roch basis computations. Our implementation bears this out, and this idea forms the basis of Algorithm 1, our improved algorithm for solving `HR-Min`. The key idea of this algorithm is to try the most likely values for r first. A formal proof of correctness is given in Theorem 1.

The approach for computing the Hess-reduction of a divisor outlined above relies on the ability to compute the Riemann-Roch dimension of a divisor E. In practice, this is done by computing a basis of $\mathcal{L}(E)$ and returning the number of basis elements. In our setting, by the monotonicity of the Riemann-Roch dimension (Lemma 1), it suffices to only compute one non-zero element of $\mathcal{L}(E)$ rather than a full basis, which may be faster. An easy modification of the Riemann-Roch basis algorithm from [7] effects this. This variant is called a *short-circuited Riemann-Roch* operation, denoted by SCRR [14, Definition 3.6.2]. This operation takes as input a divisor $E \in \mathrm{Div}(F)$ and outputs a non-zero element of

$\mathcal{L}(E)$ if $\ell(E) > 0$, or the string `None` if $\mathcal{L}(E) = \{0\}$. Note that if $\ell(E) = 1$, then SCRR(E) returns a basis of $\mathcal{L}(E)$, and div(SCRR(E)) is unique.

Algorithm 1. Optimized Hess reduction strategy via linear search

Input: $D \in \mathrm{Div}^0(F)$, $A \in \mathbb{P}(F)$ with $\deg(A) = 1$
Output: (r, a) where r solves `HR-Min` and $0 \neq a \in \mathcal{L}(D + rA)$

1: $\tilde{D} \leftarrow D + (g-1)A$
2: $a \leftarrow \mathrm{SCRR}(\tilde{D})$
3: **if** $a =$ `None` **then** ▷ If $\ell(D + (g-1)A) = 0$ then $\ell(D + gA) = 1$.
4: $\quad \tilde{D} \leftarrow \tilde{D} + A$ ▷ $\tilde{D} = D + gA$
5: $\quad a \leftarrow \mathrm{SCRR}(\tilde{D})$ ▷ $a \in \mathcal{L}(D + gA)$
6: $\quad$ **return** g, a
7: **end if**
8: **for** $m \leftarrow g-2, \ldots, 0$ **do**
9: $\quad \tilde{D} \leftarrow \tilde{D} - A$ ▷ $\tilde{D} = D + mA$
10: $\quad a' \leftarrow \mathrm{SCRR}(\tilde{D})$
11: $\quad$ **if** $a' =$ `None` **then** ▷ $\ell(D + (m+1)A) = 1$
12: $\quad\quad$ **return** $m + 1, a$ ▷ $a \in \mathcal{L}(D + (m+1)A)$
13: $\quad$ **end if**
14: $\quad a \leftarrow a'$
15: **end for**
16: **return** $0, a$

Theorem 1. *Let F/K be a function field of genus $g \geq 2$. Given $D \in \mathrm{Div}^0(F)$, Algorithm 1 returns (r, a) where r solves `HR-Min` for D and $0 \neq a \in \mathcal{L}(D + rA)$.*

Proof. Let $D \in \mathrm{Div}^0(F)$. Suppose first that Algorithm 1 returns from Line 6. Then $\ell(D+(g-1)A) = 0$. By [7, Proposition 8.2], `HR-Min` has a solution, and by the monotonicity of Riemann-Roch spaces (Lemma 1), we have $\ell(D + gA) = 1$ in order for this solution to exist. Therefore $r = g$ is the solution to `HR-Min`, and since $a \in \mathcal{L}(D + gA)$, the output (r, a) of Algorithm 1 is correct.

Next, assume that Algorithm 1 returns from Line 12. In order for this line to be reached, we must have $\ell(D+(g-1)A) \geq 1$. By monotonicity $\ell(D+gA) \geq 1$, so $r = g$ does not solve `HR-Min`. Let m be the value of the loop variable at the point where Algorithm 1 reaches Line 12. Then $\ell(D + mA) = 0$, so by monotonicity, $\ell(D + m'A) = 0$ for all $m' < m$. Since the condition in Line 11 is not satisfied for $m + 1$, we have $\ell(D + (m+1)A) \geq 1$. By monotonicity and the existence of a minimal solution, $r = m + 1$ is the minimal integer that solves `HR-Min`. As $a \in \mathcal{L}(D + (m+1)A)$, Algorithm 1 returns the correct value on Line 12.

Finally suppose that Algorithm 1 returns from Line 16. For this point to be reached we must have $\ell(D + mA) \geq 0$ for all $0 \leq m \leq g$. Since a solution to `HR-Min` with $0 \leq r \leq g$ exists, it follows that $r = 0$ must be the solution. Accordingly, our algorithm returns $r = 0$ as the solution to `HR-Min` in this case. Furthermore, since $g \geq 2$ by assumption, the for loop will run at least once, so we have $a \in \mathcal{L}(D)$. □

In the most likely case where $r = g$ solves `HR-Min`, Algorithm 1 completes with only two invocations of SCRR, in contrast to binary search which requires $O(\log g)$ such calls. We also note that the algorithm handles the second most likely case where $r = g - 1$ is the solution with the same number of steps as the $r = g$ case, so the optimal scenario occurs with expected probability $1 - q^{-2}$.

Our main strategy for designing Algorithm 1 was to minimize the number of costly calls to SCRR. The remainder of the work is given by the divisor additions in Lines 1, 4 and 9. In practice, software implementations such as SageMath [22] represent divisors as pairs of fractional ideals of the finite and infinite maximal orders, as suggested in [7]. With this representation, we are able to effect an additional performance gain at the cost of a reasonable increase to memory usage by exploiting caching, which we will describe next.

4.2 Caching

About half of the work of traditional generic Jacobian arithmetic entails arithmetic on the at most n infinite places of F. When A is chosen to be a degree one infinite place, caching intermediate results involving these places leads to effective speed-ups in Algorithm 1, at a modest increase in memory cost. For instance, cached values can be used in Lines 1, 4 and 9. The approach still works when A is a finite place of degree one, but offers fewer opportunities for caching.

For function fields with a unique infinite place of degree one, arithmetic on degree zero divisors is entirely governed by arithmetic on their finite part. Therefore, we assume henceforth that F/K is a function field of genus g with exactly $t \geq 2$ infinite places, denoted $\infty_1, \ldots, \infty_t$, with $\deg(\infty_t) = 1$. Let $D_1, D_2 \in \mathrm{Div}^0(F)$ be Hess-reduced along ∞_t, so $D_1 = \tilde{D}_1 - r_1\infty_t$ and $D_2 = \tilde{D}_2 - r_2\infty_t$ with $0 \leq r_1, r_2 \leq g$. Let $D_3 = D_1 + D_2$; we need to compute $D_3^0 = D_1^0 + D_2^0$ and $D_3^\infty = D_1^\infty + D_2^\infty$. We call the process of computing either D_3^0 or D_1^∞ a *partial divisor addition*. Since D_1 and D_2 are Hess-reduced, the number of possibilities for D_3^∞ is effectively bounded, suggesting that caching is worthwhile. A lengthy but elementary combinatorial derivation (see [14, Subsection 4.5.3]) yields a loose upper bound of $O(g^{t-1})$ ideals of $\mathfrak{O}_{F,\infty}$ on the cache for sums of infinite parts of Hess-reduced divisors. A similar technique can be applied to one of the steps in SCRR, with a memory cost of $O(g^{2t+1})$. Here, we cache a map from $\mathfrak{O}_{F,\infty}$-ideals to $n \times n$ matrices whose entries are rational functions over K of numerator and denominator degree at most g.

Empirically, the actual number of cached elements is substantially smaller than these bounds. In our experiments, the size of the caches was never an issue, and was inconsequential over larger finite fields. The typical case where $r = g$ solves `HR-Min` happens less frequently for smaller finite fields, causing more distinct intermediate values to be cached. For $g = 25$, $t = 2$, and $\mathbb{F}_{32\,771}$, after approximately 5000 Jacobian additions, the observed cache size was 60 for the addition of the infinite parts of divisors and 33 for the SCRR cache. See [14, Table 4.2] for a table of observed cache sizes.

Beyond caching, the remaining memory cost of Algorithm 2 is simply the memory cost of the input and output divisors, along with some intermediate expressions of similar size, so we do not include a full memory cost analysis.

5 Time Complexity of Computing Hess-Reductions

We now present a time complexity analysis of Algorithm 1, with and without caching, and compare it to the time complexity for the binary search approach suggested in [7]. To ensure a fair comparison, we also give the time complexity for the binary search with and without using our caching technique. We do not consider the aforementioned impact of caching on the subroutine SCRR, as it only speeds up one small part of the SCRR computation.

An appropriate measure of the size of a divisor is given by its height, which will be used in our complexity statements.

Definition 3. *[7, p. 434] Let $D \in \mathrm{Div}(F)$. We define the* (divisor) height *of D to be*

$$\mathrm{h}(D) \coloneqq \sum_{P \in \mathbb{P}(F)} |v_P(D)| \cdot \deg(P).$$

Note that $\mathrm{h}(D) = \deg(D)$ when $D \geq 0$. We express our running times in terms of the costs of Riemann-Roch basis computations and divisor additions, using the following notation:

Definition 4. *[14, Notation 4.3.3] Denote by $\mathbf{RR}(h)$ the time complexity, expressed in field operations in K, of a short-circuited Riemann-Roch computation on input a divisor $D \in \mathrm{Div}(F)$ with $h = \mathrm{h}(D)$.*

Let $D_1, D_2 \in \mathrm{Div}^0(F)$ be Hess-reduced. Denote by $\mathbf{I}$ the time complexity, expressed in field operations in K, of computing either D_3^0 or D_3^∞, where $D_3 = D_1 + D_2$.

5.1 Time Complexity in High-Level Operations

Theorem 2. *Let $D = \bar{D} - \bar{r}A$ be the unreduced sum of two Hess-reduced divisors of F. Assume that $(g-1)A$ and $-A$ have both been precomputed. Then the following hold:*

1. *In the worst case, the complexity of Algorithm 1 on input D is*

$$g\mathbf{I} + \sum_{m=0}^{g-1} \mathbf{RR}(\mathrm{h}(D + mA)). \tag{2}$$

2. *In the heuristically average case, the complexity of Algorithm 1 on input D is*

$$2\mathbf{I} + \mathbf{RR}(3g+1) + \mathbf{RR}(3g). \tag{3}$$

Proof. The worst case complexity follows from the design of the algorithm. For the heuristically average case, let D_1 and D_2 be the two Hess-reduced divisors such that $D = D_1 + D_2$. Assume that $r = g$ solves `HR-Min`, and that $D_1 = \tilde{D}_1 - r_1 A$ and $D_2 = \tilde{D}_2 - r_2 A$ are also both typical divisors, hence $r_1 = r_2 = g$. We may write $\bar{D} = \tilde{D}_1 + \tilde{D}_2$ and $\bar{r} = 2g$, so $\deg(\bar{D}) = 2g$. Algorithm 1 computes the divisor sums $D + (g-1)A$ and $D + (g-1)A + A = D + gA$, giving $2\mathbf{I}$. Algorithm 1 also computes $\mathrm{SCRR}(D + (g-1)A)$ and $\mathrm{SCRR}(D + gA)$. Noting that $\bar{D} \geq 0$ and that $A \notin \mathrm{supp}(\bar{D})$ by Proposition 1, the heights of these divisors are

$$\begin{aligned} \mathrm{h}(D + (g-1)A) &= \mathrm{h}(\bar{D}) + \mathrm{h}(-2gA + (g-1)A) = 3g + 1, \\ \mathrm{h}(D + gA) &= \mathrm{h}(\bar{D}) + \mathrm{h}(-2gA + gA) = 3g, \end{aligned}$$

whence the heuristically average case complexity follows. □

For completeness, we give the analogous result for computing Hess-reductions using binary search for solving `HR-Min`. In the typical case where $r_1 = r_2 = r = g$, this complexity is worse than that of Theorem 2 for $g \geq 4$.

Theorem 3. *Let $D = \bar{D} - \bar{r}A$ be the unreduced sum of two Hess-reduced divisors. Assume that mA has been precomputed for all $0 \leq m \leq g$. Then both the worst case and heuristically average case complexity of finding the solution r to `HR-Min` on input D with binary search and computing $a \in \mathcal{L}(D + rA)$ is*

$$\lceil \log_2(g+1) \rceil \mathbf{I} + \sum_{i=1}^{\lceil \log_2(g+1) \rceil} \mathbf{RR}\left(\mathrm{h}\left(D + \left\lceil \frac{g\left(2^i - 1\right)}{2^i} \right\rceil A\right)\right).$$

Proof. In the worst case for binary search, the sought value lies at an endpoint of the search range, which holds in the heuristically average case where $r = g$ solves `HR-Min`. Here binary search needs to compute $D + mA$ and $\mathrm{SCRR}(D + mA)$ for all $m = \left\lceil g \cdot \frac{2^i - 1}{2^i} \right\rceil$ for all $1 \leq i \leq \lceil \log_2(g+1) \rceil$. □

5.2 Time Complexity in Base Field Operations

To give the time complexity in base field operations, we must first fix a representation of divisors, then substitute the best known values for the two subroutines $\mathbf{I}$ and $\mathbf{RR}(h)$ into Theorems 2 and 3.

In practice, divisors are represented by a pair (I, J) where I and J are ideals of $\mathfrak{O}_{F,0}$ and $\mathfrak{O}_{F,\infty}$, respectively. They are represented as $n \times n$ matrices in Hermite Normal Form whose entries are polynomials in $K[x]$ of degree at most g when the divisor is Hess-reduced.

The cost of $\mathbf{I}$ is simply the complexity of multiplying two ideals and converting the resulting matrix to Hermite Normal Form. The best asymptotic complexity for $\mathbf{I}$ that we are aware of arises from replacing naive polynomial multiplication with FFT in [20, Theorem 5.2.2], yielding

$$\mathbf{I} = O(n^6(g \log(gn) \log\log(gn) + g^2)). \tag{4}$$

To the best of our knowledge, the fastest asymptotic complexity for $\mathbf{RR}(h)$ can be found in [1, Corollary 4.1.5] and is given by

$$\mathbf{RR}(h) = O(n^5(h + n^2 C_f)^2). \tag{5}$$

In practice, the performance is better than predicted by Eqs. (4) and (5), as seen in Figs. 1 and 2.

Jacobian arithmetic with Hess-reduced divisors is performed using Algorithm 2. So the complexity of Jacobian arithmetic is simply the complexity of Algorithm 2 with whichever algorithm is used to compute (r, a) in Line 2.

Algorithm 2. Jacobian arithmetic with Hess-reduced divisors

Input: Hess-reduced divisors $D_1 = \tilde{D}_1 - r_1 A$ and $D_2 = \tilde{D}_2 - r_2 A$
Output: $D_3 = \tilde{D}_3 - r_3 A$ with D_3 Hess-reduced and $D_3 \equiv D_1 + D_2$
1: $E \leftarrow D_1 + D_2$
2: Compute (r, a) where r is a solution to `HR-Min` for input E and $a \in \mathcal{L}(D + rA) \setminus \{0\}$
3: $D_3 \leftarrow E + \mathrm{div}(a)$

We can now give the time complexity of Jacobian arithmetic with Algorithm 1 or with binary search, both with and without caching. We account for the impact of caching by assuming that adding the infinite parts of divisors is amortized as $O(1)$.

Theorem 4. *Computing the Hess-reduction of two Hess-reduced divisors of F has the following asymptotic time complexity, expressed in operations in K.*

1. *Using Algorithm 1 in Line 2 of Algorithm 2 and no caching, the worst case time complexity is*
$$O(C_f^2 g n^9 + C_f g^2 n^7 + g^3 n^6)$$
and the heuristically average case complexity is
$$O(C_f^2 n^9 + C_f g n^7 + g^2 n^6 + g n^6 \log(gn) \log\log(gn))$$
2. *Using Algorithm 1 in Line 2 of Algorithm 2 and caching, the worst case time complexity is*
$$O(C_f^2 g n^9 + C_f g^2 n^7 + g^3 n^5)$$
and the heuristically average case complexity is
$$O(C_f^2 n^9 + C_f g n^7 + g^2 n^6 + g n^6 \log(gn) \log\log(gn))$$
3. *Using binary search in Line 2 of Algorithm 2 and no caching, the time complexity is*
$$O(\log(g)(C_f^2 n^9 + C_f g n^7 + g^2 n^6 + g n^6 \log(gn) \log\log(gn)))$$

4. *Using binary search in Line 2 of Algorithm 2 and caching, the time complexity is*

$$O(\log(g)(C_f^2 n^9 + C_f g n^7 + g^2 n^5) + g^2 n^6 + g n^6 \log(gn) \log\log(gn))$$

Proof. In Algorithm 2, we add the finite and infinite parts of D_1 and D_2 to compute E, for a cost of $2\mathbf{I}$. One of these $\mathbf{I}$ operations is on the infinite parts of D_1 and D_2, hence amortized as $O(1)$ if caching is used. We then use either Algorithm 1 or binary search to find (r, a), with the corresponding costs given in Theorems 2 and 3. Finally, we compute $D_3 = E + \mathrm{div}(a)$ for a cost of $2\mathbf{I}$, and again one of these is $O(1)$ if caching is used. The computation of $\mathrm{div}(a)$ from a is dominated by the other costs in the algorithm. After accounting for caching in Algorithm 1 and in the binary search approach and substituting Eqs. (4) and (5) into Theorems 2 and 3, we obtain the asserted complexity estimates. □

We determine the overall asymptotic complexities when one of g, n is fixed and the other varies. Although caching makes a difference in practical performance, after fixing either g or n and allowing the other to vary (see Figs. 1 and 2), it only impacts lower order terms in the overall complexities.

Corollary 1. *Computing the Hess-reduction of two Hess-reduced divisors of F has the following asymptotic time complexity, expressed in operations in K.*
When g is fixed and $n \to \infty$: $O(C_f^2 n^9)$
When n is fixed and $g \to \infty$:

(a) $O(C_f^2 g)$ in the worst case when using Algorithm 1 in Line 2 of Algorithm 2;
(b) $O(C_f^2)$ heuristiclly on average when using Algorithm 1 in Line 2 of Algorithm 2;
(c) $O(C_f^2 \log(g))$ when using binary search in Line 2 of Algorithm 2.

Proof. For fixed g and varying n, the term $C_f^2 n^9$ dominates in all the expressions of Theorem 4. For fixed n and varying g, we use the identity

$$g \leq \frac{1}{2}(C_f n - 2)(n - 1); \tag{6}$$

established in [7, Cor. 5.6] and identify the dominant term in each of the complexity estimates of Theorem 4. □

Asymptotically, for fixed n and varying g, the heuristic average complexities of using linear versus binary search in Algorithm 2 differ by a factor of $O(\log(g))$. Indeed, for typical Hess-reduced input divisors, Algorithm 1 will complete after 2 reductions, whereas binary search is expected to perform $\lceil \log_2(g+1) \rceil$ reductions, yielding a relative factor of $\lceil \log_2(g+1) \rceil / 2$.

6 Timing Experiments

We implemented Jacobian arithmetic on Unique Hess representations in Sage using both Algorithm 1 and binary search for reduction, both with and without caching, and profiled the results. Our implementation of Unique Hess arithmetic using Algorithm 1 was accepted into SageMath, available since the SageMath 10.9.beta7 release [22]. Our implementation of the binary search approach as well as our code used to profile the implementations is available at [15]. Our implementations of Algorithm 1 and the binary search variant were mostly written in Python, with some performance-critical subroutines written in Cython.

We carried out our timing experiments on a server running Red Hat Enterprise Linux 9.7 with an Intel Xeon CPU E7-8891 v4 with 80 64-bit CPU cores at 2.80 GHz, and 256 GiB of memory. Our experiments were run on a custom fork of Sage 10.7.beta7, using Python 3.12.5. SageMath has over 100 software dependencies, so we will only list major relevant dependencies: Singular 4.4.1 [24], FLINT 3.4.0 [21], GMP 6.3.0 [5], and numpy 2.3.2 [6]. We also used Magma 2.27–6 [2] to perform some genus computations that took too long in SageMath. Our use of Magma was restricted to computing genera when initializing some function fields, and so it has no impact on our reported timing results.

6.1 Experimental Setup

For testing, we generated global function fields both via an ad-hoc method and via the method described in [20]. The latter provides an easy way to generate global function fields with two infinite places of degree one. These function fields are given by a defining polynomial $f(t) \in K[x,t]$ of degree $n = [F : K(x)]$, irreducible over $K(x)$, and of the form:

$$f(t) = t^n + \sum_{i=0}^{n-1} a_i t^i,$$

with $a_i \in K[x]$ for $0 \leq i \leq n-1$, and

$$\begin{aligned} \deg(a_{n-1}) &= C_f, \\ \deg(a_i) &< (n-i)C_f \quad \text{for } 1 \leq i \leq n-1, \\ \deg(a_0) &= nm - 1. \end{aligned}$$

We conjecture that for these function fields, equality always holds in (6). We verified this for all the fields we tested for producing Fig. 1, so our timing results do not depend on this assumption. Note that equality in (6) implies that any two of g, n and C_f uniquely determine the third.

For Fig. 2, we instead generated function fields via an ad-hoc method, by trying random polynomials $f(x,t)$, monic and of degree 3 in t, with appropriate bounds on C_f, verifying that they were irreducible over $K(x)$ and that the corresponding function field contained a degree one infinite place. Equality did not hold in Eq. (6) for the function fields generated by this ad-hoc method.

Due to space constraints, we only report timing tests over the finite field $\mathbb{F}_{32771}$, a 16 bit prime field. In [14, Chapter 5] the first author conducted extensive timing experiments on a wide range of parameter sets. The data presented here is representative of the findings in that source, with the possible exception of very small primes like $p = 2, 3$ where the generic case for which Algorithm 1 is optimized does not happen sufficiently frequently.

For each triple (g, n, p), we performed tests in 5 function fields. In each function field we performed Unique Hess arithmetic on 5 Fibonacci-style addition chains of length 1000. Reported results are the average over all function fields and addition chains for each (g, n, p).

To test the correctness of our implementations, we relied on SageMath's robust testing framework that is able to automatically create a test suite and check correctness by verifying that the abelian group axioms hold on a set of examples.

6.2 Timing Results

Overall, Algorithm 1 significantly outperformed the binary search approach. In our figures, we plotted the average CPU time in milliseconds to perform a single Jacobian addition for various parameters and implementations, alongside the corresponding asymptotic complexities. For Algorithm 1 we used the heuristically average case complexity. For the asymptotic complexity plots in Fig. 1, we assumed equality in Eq. (6) and replaced C_f accordingly. In Fig. 2 we treated C_f as a constant because C_f^2 is negligible compared to n^9 asymptotically, and the largest value of C_f in our data plotted in Fig. 2 was $C_f = 6$. To account for constant factors in the plots of our asymptotic estimates, we computed a constant a such that if the complexity is $O(f(x))$, then $af(x)$ matches the data at a single measured value. For Fig. 1, since our improvement is by a factor of $O(\log(g))$, we matched the constant a to the largest data point $g = 55$ in order to more accurately capture the dependence on g. In Fig. 2, we matched the constant a to the first data point as our improvements do not affect the asymptotic complexity in n.

Figure 1 shows that caching effects a noticeable speed-up, but the most significant performance gains come from using Algorithm 1. In particular, for $g \geq 13$, the non-caching implementation using Algorithm 1 outperforms the caching implementation using binary search. As predicted by our complexity analysis, the difference in performance between Algorithm 1 and the binary search approach grows with g. Our data show an increase in speed that is slightly less than the predicted factor of $\lceil \log(g+1) \rceil / 2$ for linear versus binary search, likely because this speed-up only applies in the most expensive part of Jacobian arithmetic (reduction to a unique representative), and the lower degree terms that were excluded from the analysis in Corollary 1 are not entirely negligible for our parameter range. For genus 100 (not depicted in Fig. 1), the linear search variant was about 3 times faster both with and without caching. Here, we would expect the performance to increase by a factor of about $\lceil \log_2(101) \rceil / 2 = 3.5$, which is reasonably close to our measured speed-up of a factor of 3. The relative impact

of caching is smaller as the genus grows, since the computations that are cached take up a smaller proportion of the total time with increasing genus. For the linear search variant, caching gave a speedup of a factor of 1.63 for genus 4, was still noticeable at 1.15 for genus 25, but was negligible at 1.01 for genus 100.

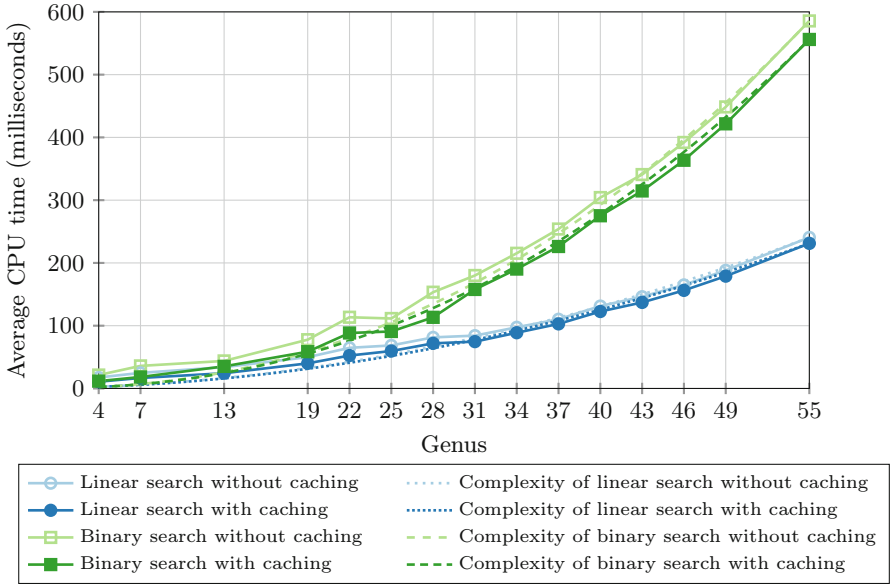

Fig. 1. Addition performance for $4 \leq g \leq 55$, $n = 3$, $p = 32771$.

For fixed g and varying n, all our algorithms have same the asymptotic complexity by Corollary 1. With g fixed, caching effects a speed-up by a constant factor in the number of partial divisor additions, with a more pronounced performance gain for the computationally more intense binary search compared to Algorithm 1. However, this constant factor speed-up is dominated by the cost of the short-circuited Riemann-Roch computations. Linear search is expected to be faster on average than binary search by a constant factor of approximately $\lceil \log_2(g+1) \rceil / 2$ as explained earlier; for $g = 15$, this factor is 2. The data depicted in Fig. 2 bear this out and also suggest that the asymptotic upper bound of $O(C_f^2 n^9)$ is an over-estimate of the actual complexity of Jacobian arithmetic. For $g = 15$, caching produced a speedup of a factor ranging from 1.20 to 1.30, without any strong trends dependent on n.

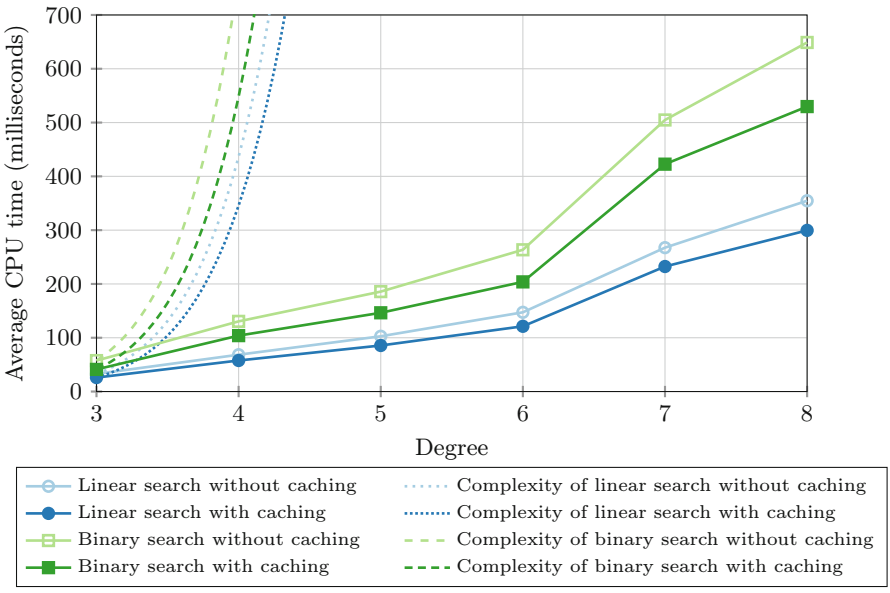

Fig. 2. Addition performance for $3 \leq n \leq 8$, $g = 15$, $p = 32771$.

7 Conclusions and Further Research

Our reduction algorithm (Algorithm 1) was designed specifically for optimal performance in the most frequent cases. Further speed-ups will likely require improvements to the underlying subroutines, namely Riemann-Roch space computation and ideal multiplication. Additional information about the input divisor might potentially narrow down the possible candidate solutions for `HR-Min`, but given the overwhelming likelihood of the generic case, it is difficult to see how that approach would lead to a significant performance gain in this situation. Even *a priori* ruling out the case where $r = g - 1$ solves `HR-Min` without performing a short-circuited Riemann-Roch computation would only speed up the algorithm by a constant factor.

The timing experiments of Sect. 6 demonstrate that Algorithm 1 is the fastest unique divisor reduction algorithm in practice, at least for the function fields we tested. As the non-caching version of the linear search implementation outperformed the caching version of the binary search implementation, we do not need to assume the existence of a degree one infinite place for the linear search method to outperform the binary search approach. Our timing experiments also demonstrate that caching computations involving the infinite parts of divisors gives a noticeable speedup when the function field has a degree one infinite place.

7.1 Future Work

We list several possible directions for future work.

For short-circuited Riemann-Roch computations, we used SageMath's modified version of the algorithm for computing K-bases of Riemann-Roch spaces from [7]. Algorithm 1 demonstrates that short-circuited Riemann-Roch computations are sufficient for some applications. It would be beneficial to have a potential SCRR approach that is faster than simply exiting early from an algorithm designed to perform a full Riemann-Roch basis computation.

For hyperelliptic curves, the fastest known algorithm for Jacobian arithmetic is NUCOMP [9,13]. The traditional approach to divisor arithmetic first computes the sum of two input divisors and subsequently reduces the result. NUCOMP in essence interleaves divisor addition and reduction, thereby operating on smaller inputs and avoiding the costly computation of bases for non-reduced intermediate divisors. It is worthwhile to explore whether a similar strategy can be realized for Jacobian arithmetic for non-hyperelliptic curves.

The first framework for arithmetic on unique representatives of elements in the Jacobian of a split hyperelliptic curve (i.e. a hyperelliptic curve with two infinite places) is due to Paulus and Rück [17]. In addition to reduction, the Paulus-Rück arithmetic requires additional adjustment steps to obtain such unique representatives. These adjustment steps were eliminated in the balanced divisor arithmetic introduced by Galbraith, Harrison, and Mireles Morales [3,4,16]. In [14], it was shown that unique reduced divisor class representatives of [17] are a special case of the Hess-reductions introduced in Definition 2 when the curve is split hyperelliptic. This relationship could potentially be leveraged to explore whether the notion of a balanced divisor could be extended to non-hyperelliptic curves with two infinite places, or even to curves with more infinite places, with the aim to simplify Jacobian arithmetic and/or improve its performance.

The notion of a semi-reduced divisor plays a crucial role in Jacobian arithmetic for hyperelliptic curves. Divisor addition produces a semi-reduced divisor that is equivalent to the sum of two input divisors, which is subsequently reduced. The concept of semi-reduced divisors was generalized to arbitrary function fields in [14]; in essence, a divisor D is semi-reduced if no sub-sum of D is the conorm of a divisor of $K(x)$. The author of [14] also proved that Hess-reductions along an infinite place of degree one are semi-reduced. Future work could explore whether the notion of semi-reducedness can be generalized further in such a way that Hess-reductions along a finite degree one place, or even along a degree one divisor as considered in [7], are semi-reduced. An appropriate notion of semi-reducedness may be a key ingredient for an efficient reduction algorithm that produces unique representatives of divisor classes.

Acknowledgments. All three authors were supported by the Natural Sciences and Engineering Research Council of Canada. The first author also received funding from the Province of Alberta through an AGES Research scholarship. We are grateful to three anonymous reviewers for their valuable comments. We also thank Kwankyu Lee for writing much of the SageMath function field machinery that we built on, and for reviewing the first author's pull request https://github.com/sagemath/sage/pull/41453 adding Unique Hess Jacobian arithmetic to SageMath.

Disclosure of Interests. The authors have no competing interests.

References

1. Bauch, J.D.: Lattices over polynomial rings and applications to function fields. Ph.D. thesis, Universitat Autònoma de Barcelona, Bellaterra (2014)
2. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system I: the user language. J. Symb. Comput. **24**(3–4), 235–265 (1997). https://doi.org/10.1006/jsco.1996.0125
3. Galbraith, S.D.: Mathematics of Public Key Cryptography. 2 edn. (2018)
4. Galbraith, S.D., Harrison, M., Mireles Morales, D.J.: Efficient hyperelliptic arithmetic using balanced representation for divisors. In: van der Poorten, A.J., Stein, A. (eds.) ANTS 2008. LNCS, vol. 5011, pp. 342–356. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79456-1_23
5. Granlund, T., et al.: GNU Multiple Precision Arithmetic Library. Free Software Foundation, Inc (2023)
6. Harris, C.R., et al.: Array programming with NumPy. Nature **585**(7825), 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2
7. Hess, F.: Computing Riemann-Roch spaces in algebraic function fields and related topics. J. Symb. Comput. **33**(4), 425–445 (2002). https://doi.org/10.1006/jsco.2001.0513
8. Howe, E., Lauter, K.E., Top, J.: Pointless curves of genus three and four. In: Arithmetic, Geometry and Coding Theory (AGCT 2003). Séminaires et Congrès, vol. 11, pp. 125–141. Société Mathématique de France, Paris (2005)
9. Jacobson, Jr., M.J., van der Poorten, A.J.: Computational aspects of NUCOMP. In: Algorithmic number theory (Sydney, 2002), Lecture Notes in Comput. Sci., vol. 2369, pp. 120–133. Springer, Berlin (2002). https://doi.org/10.1007/3-540-45455-1_10
10. Junge, M.: Asymptotically Fast Arithmetic in the Picard Group of Algebraic Curves & Related Topics. Ph.D. thesis, Universität Oldenburg (2022)
11. Khuri-Makdisi, K.: Linear algebra algorithms for divisors on an algebraic curve. Math. Comput. **73**(245), 333–357 (2004). https://doi.org/10.1090/S0025-5718-03-01567-9
12. Khuri-Makdisi, K.: Asymptotically fast group operations on Jacobians of general curves. Math. Comput. **76**(260), 2213–2239 (2007). https://doi.org/10.1090/S0025-5718-07-01989-8
13. Lindner, S., Imbert, L., Jacobson, M.J.: Balanced NUCOMP. In: Boulier, F., England, M., Sadykov, T.M., Vorozhtsov, E.V. (eds.) CASC 2020. LNCS, vol. 12291, pp. 402–420. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60026-6_23
14. Macri, V.: Comparison of and improvements to degree zero divisor class group arithmetic in algebraic function fields. Master's thesis, University of Calgary (2025). https://doi.org/10.11575/PRISM/50422
15. Macri, V.: Improvements to Jacobian arithmetic in global function fields (2026). https://github.com/vincentmacri/Improvements-to-Jacobian-Arithmetic-in-Global-Function-Fields
16. Mireles Morales, D.J.: Efficient Arithmetic on Hyperelliptic Curves with Real Representation. Ph.D. thesis, The University of London (2008)

17. Paulus, S., Rück, H.G.: Real and imaginary quadratic representations of hyperelliptic function fields. Math. Comput. **68**(227), 1233–1241 (1999). https://doi.org/10.1090/S0025-5718-99-01066-2
18. Stichtenoth, H.: Algebraic Function Fields and Codes, Graduate Texts in Mathematics, 2 edn., vol. 254. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-540-76878-4
19. Sutherland, A.V.: Fast Jacobian arithmetic for hyperelliptic curves of genus 3. Open Book Ser. **2**(1), 425–442 (2019). https://doi.org/10.2140/obs.2019.2.425
20. Tang, A.: Infrastructure of function fields of unit rank one. Ph.D. thesis, University of Calgary (2011)
21. The FLINT team: FLINT: Fast Library for Number Theory (2025)
22. The Sage Developers: SageMath, the Sage Mathematics Software System (2026)
23. Wiles, A.: The Birch and Swinnerton-Dyer conjecture
24. Wolfram, D., Greuel, G.M., Pfister, G., Schönemann, H.: Singular 4-4-1 — a computer algebra system for polynomial computations (2025)