Balanced NUCOMP

Sebastian Lindner, Laurent Imbert, Michael Jacobson Jr

Balanced NUCOMP

Sebastian Lindner[1] Laurent Imbert[2] Michael J. Jacobson, Jr.[1] *

[1] Department of Computer Science, University of Calgary, Calgary, Canada
{salindne,jacobs}@ucalgary.ca
[2] CNRS, LIRMM, Universit de Montpellier, Montpellier, France
laurent.imbert@lirmm.fr

Abstract. Arithmetic in the divisor class group of a hyperelliptic curve is a fundamental component of algebraic geometry packages implemented in computer algebra systems such as Magma and Sage. In this paper, we present an adaptation of Shanks' NUCOMP algorithm for split model hyperelliptic curves of arbitrary genus that uses balanced divisors and includes a number of enhancements to optimize its efficiency in that setting. Our version of NUCOMP offers better performance than Cantor's algorithm in the balanced divisor setting. Compared with Magma's built-in arithmetic, our Magma implementation shows significant speed-ups for curves of all but the smallest genera, with the improvement increasing as the genus grows.

1 Introduction

The divisor class group of a hyperelliptic curve defined over a finite field is a finite abelian group at the center of a number of important open questions in algebraic geometry and number theory. Sutherland [14] surveys some of these, including the computation of the associated L-functions and zeta functions used in his investigation of Sato-Tate distributions [13]. Many of these problems lend themselves to numerical investigation, and as emphasized by Sutherland, fast arithmetic in the divisor class group is crucial for their efficiency. Indeed, implementations of these fundamental operations are at the core of the algebraic geometry packages of widely-used computer algebra systems such as Magma and Sage.

All hyperelliptic curves are represented as models that are categorized as either ramified (imaginary), split (real), or inert according to their number of points at infinity defined over the base field. Ramified curves have one point at infinity, whereas split curves have two. Inert (also called unusual) curves have no infinite points defined over the base field and are usually avoided in practice as they have cumbersome divisor class group arithmetic and can be transformed to a split model over at most a quadratic extension of the base field.

Divisor class group arithmetic differs on ramified and split models. The split scenario is more complicated. As a result, optimizing divisor arithmetic on split

* The third author is supported in part by NSERC of Canada.

hyperelliptic curves has received less attention from the research community. However, split models have many interesting properties; most importantly, there exists a large array of hyperelliptic curves that cannot be represented with a ramified model and require a split model representation. Thus, exhaustive computations such as those in [13] conduct the bulk of their work on split models by necessity.

Arithmetic in the divisor class group of a hyperelliptic curve can be described algebraically using an algorithm due to Cantor [1], and expressed in terms of polynomial arithmetic. Various improvements and extensions to Cantor's algorithm have been proposed for ramified model curves, including an adaptation of Shank's NUCOMP algorithm [11] for composing binary quadratic forms [6]. The main idea behind NUCOMP is that instead of composing two divisors directly and then reducing to find an equivalent reduced divisor, a type of reduction is applied part way through the composition, so that when the composition is finished the result is almost always reduced. The effect is that the sizes of the intermediate operands are reduced, resulting in better performance in most cases. Improvements to NUCOMP have been proposed, most recently the work of [5], where best practices for computing Cantor's algorithm and NUCOMP are empirically investigated.

NUCOMP has also been proposed for arithmetic in the so-called infrastructure of a split model curve [8]. However, as shown by Galbraith et. al. [4,9], arithmetic on split model hyperelliptic curves is most efficiently realized via a divisor arithmetic framework referred to as balanced. Although the balanced and the infrastructure frameworks are similar, NUCOMP had yet to be applied explicitly to the former.

In this paper, we present an adaptation of NUCOMP for divisor class group arithmetic on split model hyperelliptic curves in the balanced divisor framework. We incorporate optimizations from previous works in the ramified model setting and introduce new balanced setting-specific improvements that further enhance practical performance. Specifically, our version of NUCOMP includes various improvements over its infrastructure counterpart [8]:

- it describes for the first time exactly how to use NUCOMP in the framework of balanced divisors, including explicit computations of the required balancing coefficients;
- it introduces a novel normalization of divisors in order to eliminate the extra adjustment step required in [8] for typical inputs when the genus of the hyperelliptic curve is odd, so that in all cases typical inputs require no extra reduction nor adjustment steps;
- it uses certain aspects of NUCOMP to compute one adjustment step almost for free in some cases.

We present empirical results that demonstrate the efficiency gains realized from our new version of NUCOMP as compared with the previous best balanced divisor class group arithmetic based on Cantor's algorithm and the arithmetic implemented in Magma, showing that NUCOMP is the method of choice for all but the smallest genera. With our improvements, NUCOMP is more efficient

than Cantor's algorithm for genus as low as 5, compared to 7 using the version in [8], both of which are within the possible range of applications related to numerical investigations of number-theoretic conjectures. Our implementation is faster than Magma's built-in arithmetic for $g \geq 7$, and the gap increases with the genus; we assume that a more even comparison using, for example, optimized C implementations would further narrow this gap in performance.

The rest of the paper is organized as follows. In Section 2 we provide background information on split model hyperelliptic curves. Balanced divisor arithmetic using Cantor's algorithm is presented in Section 3. In Section 4 we present our version of NUCOMP for the balanced divisor setting, as well as details of our improvements. In Section 5 we present empirical results comparing our version of NUCOMP to Cantor's algorithm and Magma's built-in arithmetic. Finally, we give some conclusions and directions for future work in Section 6.

2 Background

In this section we recall the essential relevant notions related to divisor classes of hyperelliptic curves and their arithmetic. For more details and background, the reader is referred to [10, § 12.4] for Section 2.1 and [3, Chapter 7] for Section 2.2.

2.1 Split Model Hyperelliptic Curves

As described in [10, Definition 12.4.1], a *hyperelliptic curve* C of genus g defined over a finite field k is given by a hyperelliptic equation

$$y^2 + h(x)y = f(x), \quad \text{with } h, f \in k[x],$$

that is absolutely irreducible and non-singular. A *split model* for a hyperelliptic curve C of genus g over k is given by a hyperelliptic equation satisfying $\deg(f) = 2g + 2$ and $\deg(h) \leq g + 1$. In addition, the leading coefficient of f is a square except over fields of characteristic 2 where it is of the form $s^2 + s$ for some $s \in k^*$.

Let $C(\overline{k})$ be the set of $\overline{k}$-rational points of C. The *hyperelliptic involution* of C is the map $\iota : C(\overline{k}) \to C(\overline{k})$ that sends a finite point $P = (x, y)$ on C to the point $\overline{P} = \iota(P) = (x, -y - h(x))$ on C. A point P on C is *ramified* if $\iota(P) = P$, and unramified otherwise.

The model used to represent a hyperelliptic curve determines the number and type of points at infinity. A split model representation has two unramified k-rational points at infinity denoted ∞^+ and ∞^-, where $\iota(\infty^+) = \infty^-$. Ramified models have a single ramified k-rational point at infinity, and inert models have none. It is sometimes possible to change the model of a curve C without modifying the field of definition k by translating other points to infinity. If C has a ramified k-rational point, one can obtain a ramified model for C by translating this point to infinity, by [10, Theorem 12.4.12]. If C does not have a ramified k-rational point, but has an unramified k-rational point, then similarly, that point can be translated to infinity, providing two points at infinity ∞^+, ∞^- and thus

C can be represented with a split model. If no k-rational points exist, including at infinity, then the hyperelliptic curve C can only be represented by an inert model; no alternative ramified or split models are possible over k. However, hyperelliptic curves that have neither a ramified nor an unramified k-rational point are rare and only exist over fields whose cardinality is small relative to the genus. If k is a finite field of cardinality q, the Weil bound $\#C(k) \geq q+1-2g\sqrt{q}$ guarantees that a genus g hyperelliptic curve C over k has a k-rational point whenever $q > 4g^2$, and an unramified k-rational point when $q > 4g^2 + 2g + g$.

Split models therefore are more general than ramified, as ramified models are only obtainable when the curve has a ramified point. Inert models of curves can easily be avoided in practice by translating to a split or ramified model when q is sufficiently large to guarantee a k-rational point, or by considering the curve as a split model over a quadratic extension of k otherwise. Thus, in this work we only consider improvements for hyperelliptic curves given by a split model, with performance comparisons to ramified models given in Section 5.

2.2 Divisor Class Groups of Split Model Hyperelliptic Curves

A *divisor* on a hyperelliptic curve C defined over k is a formal sum $D = \sum n_P P$ of points $P \in C(\overline{k})$ with only finitely many $n_P \neq 0$. The *support* of D, denoted $\mathrm{supp}(D)$, is the set of points $P \in C(\overline{k})$ occurring in D with $n_P \neq 0$. The *degree* of D is $\deg(D) = \sum n_p$. A divisor D is said to be *defined over* k if $\sigma D = \sum n_P \sigma P = D$ for all $\sigma \in \mathrm{Gal}(\overline{k}/k)$. The set of all degree zero divisors on C defined over k, denoted $\mathrm{Div}_k^0(C)$, is an Abelian group under component-wise addition. A divisor is *principal* if it is of the form $\mathrm{div}(\alpha) = \sum_P \mathrm{ord}_P(\alpha)P$ for some function $\alpha \in k(C)^*$ where $k(C) = k(x, y)$ is the function field of C. Principal divisors have degree zero and the set of all principal divisors $\mathrm{Prin}_k^0(C) = \{\mathrm{div}(\alpha) \mid \alpha \in k(C)^*\}$ is a subgroup of $\mathrm{Div}_k^0(C)$. The *divisor class group* of C defined over k is the quotient group $\mathrm{Pic}_k^0(C) = \mathrm{Div}_k^0(C)/\mathrm{Prin}_k^0(C)$. The principal divisor corresponding to $\infty^+ - \infty^-$ (resp. $\infty^- - \infty^+$) is denoted D_{∞^+} (resp. D_{∞^-}).

A divisor $D = \sum_P n_P P$ is *affine* if $n_{P_\infty} = 0$ for all k-rational points at infinity P_∞ on C. The divisor D is *effective* if $n_P \geq 0$ for all points P. An effective divisor can be written as $\sum P_i$, where the P_i need not be distinct. An affine effective divisor $D = \sum P_i$ is *semi-reduced* if for any $P_i \in \mathrm{supp}(D)$, $\iota(P_i) \notin \mathrm{supp}(D)$, unless $P_i = \iota(P_i)$. A semi-reduced divisor D is *reduced* if $\deg(D) \leq g$.

A semi-reduced divisor D has a compact *Mumford representation* $D = (u, v)$ such that $u, v \in k[x]$, $\deg(v) < \deg(u)$, u is monic, and $u|(v^2+vh-f)$. Explicitly, u is defined as the polynomial whose roots are the x-coordinates of every affine point in the support of $D = \sum n_P P$ accounting for multiplicity, i.e. $u = \prod_i (x - x_i)^{n_{P_i}}$ for all $P_i = (x_i, y_i) \in \mathrm{supp}(D)$. The polynomial v is the interpolating polynomial that passes through the points P_i. The Mumford representation of D is said to be reduced if $\deg(u) \leq g$. The *degree* of a semi-reduced divisor $D = (u, v)$ in Mumford representation is given by $\deg(D) = \deg(u)$.

Every rational divisor class in $\mathrm{Pic}_k^0(C)$ can be represented by a degree zero divisor $[D]$ that has a semi-reduced affine portion, but this representation is not necessarily unique. As described in [4], let D_∞ be an effective divisor of

degree g supported on k-rational points at infinity. Over split model curves, let $D_\infty = \lceil g/2 \rceil \infty^+ + \lfloor g/2 \rfloor \infty^-$. Then, every divisor class $[D]$ over genus g hyperelliptic curves described with a split model can be uniquely written as $[D_0 - D_\infty]$ where $D_0 = D_a + D_i$ is a k-rational divisor of degree g with the affine portion D_a reduced, and D_i is a specially-chosen divisor supported on the infinite points, for example as described in the next section. Note that as is standard practice, we refer to the degree of such a divisor class representative as the degree of the affine part D_a, although this is a slight abuse of notation as the divisor D technically has degree zero.

3 Balanced Divisor Arithmetic using Cantor's Algorithm

Over split model curves, divisor classes in $\mathrm{Pic}_k^0(C)$ do not have a unique, reduced Mumford representation. Mumford representation only utilizes information about the affine portion D_a of $D_0 = D_a + D_i$ for $D = D_0 - D_\infty$; uniqueness is lost because the same affine portion of D_0 could be combined with different multiples of ∞^+ and ∞^- in D_i to represent different divisor classes. Galbraith et. al. [4] defined a *reduced balanced divisor representation* for split model curves which appends to the polynomials u, v a balancing coefficient n, the number of copies of ∞^+ in D_0, hence $[D] = [u, v, n]$. In order for this representation to be unique and reduced, n is kept small, in the range $[0, g - \deg(u)]$ and $\deg u \leq g$. A divisor class $[D] = [u, v, n]$ therefore corresponds to

$$[u, v, n] = [u, v] + n\infty^+ + (g - \deg(u) - n)\infty^- - D_\infty.$$

In this notation, for example,

$$[1, 0, \lceil g/2 \rceil] = [1, 0] + \lceil g/2 \rceil \infty^+ + \lfloor g/2 \rfloor \infty^- - D_\infty$$

is the unique representative of the neutral divisor class in $\mathrm{Pic}_k^0(C)$.

Addition of divisor classes represented as reduced balanced divisors, as described in [4], is done via a two-step process. First, the affine parts of the divisors are added and reduced using Cantor's algorithm, while computing the new balancing coefficient n of the result. At this point it is possible that the resulting divisor class is neither reduced nor balanced, so a series of adjustment steps is applied, *up-adjustments* if n needs to be increased and *down-adjustments* if it needs to decrease, until the n value satisfies $0 \leq n \leq g - \deg u$ and is thus balanced. The main advantage of using balanced divisor representatives is that in the generic case, where both divisors have degree g and $n = 0$, the number of adjustment steps required is zero for even genus and one for odd genus.[3]

The two algorithms that we present in the following sections for addition and reduction (Algorithm 1) and for adjustment (Algorithm 2) follow this strategy with a variety of practical improvements. We adopt an alternative normalization

[3] The required adjustment step over odd genus reduces the degree of the intermediate divisor, similar to a reduction step.

of the v polynomial from the Mumford representation, as well as well-known algorithmic improvements to Cantor's algorithm described, for example, in [5], as described next. In all algorithms presented, let $\mathrm{lc}(a)$ denote the leading coefficient of polynomial a and $\mathrm{monic}(a) = a/\mathrm{lc}(a)$, i.e. the polynomial a made monic.

3.1 Extended Mumford Representation and Tenner's Algorithm

One standard optimization for arithmetic with ideals of quadratic number fields is to represent the ideal as a binary quadratic form, a representation that includes a third redundant coefficient that is useful computationally. In the context of divisor arithmetic, this means adding the polynomial $w = (f - v(v+h))/u$ to the Mumford representation, so that balanced divisor classes in our implementation have four coordinates, $[u, v, w, n]$.

This polynomial must be computed in every application of Cantor's algorithm as well as in reduction steps and adjustment steps. Having it available as part of the divisor representation results in some savings in the divisor addition part, and allows for the use of Tenner's algorithm for reduction and adjustment steps, a standard optimization for computing continued fraction expansions of quadratic irrationalities (see, for example, [7, §3.4]).

3.2 Divisor Representation using Reduced Bases

The standard Mumford representation of a divisor $[u, v]$ has v reduced modulo u, but any other polynomial congruent to v modulo u can also be used. In split model curves, an alternate representation called the *reduced basis* turns out to be computationally superior in practice. Reduced bases are defined in terms of the unique polynomial V^+, the principal (polynomial) part of the root y of $y(y + h(x)) - f(x) = 0$ for which $\deg(f - V^+(V^+ + h)) \leq g$, or the other root $V^- = -V^+ - h$. Note that such V^+ and V^- only exist for split models.

We say that a representation of the affine divisor $[u, v]$ given by $[u, \tilde{v}]$ is in *reduced basis* or *positive reduced basis* if $\tilde{v} = V^+ - [(V^+ - v) \pmod u]$ and in *negative reduced basis* if $\tilde{v} = V^- - [(V^- - v)) \pmod u]$. To convert a divisor $[u, v, w, n]$ into negative reduced basis $[u, v', w', n]$, first compute $(q, r) = \mathrm{DivRem}(V^- - v, u)$, where we define $\mathrm{DivRem}(a, b)$ as q, r, the quotient and remainder, respectively, obtained when dividing a by b, i.e. $a = qb + r$. For uniqueness, we take the remainder r satisfying $\deg(r) < \deg(b)$. Then $\hat{v} = V^- - r$, $w' = w - q(v + h + \hat{v})$, and let $v' = \hat{v}$. To convert back to positive reduced basis, first Compute $q \neq 0$, r such that $v' = qu + r$, then $w = w' - q(v' + h + r)$, and let $v = r$.

In both types of reduced basis, cancellations cause the degree of $f - \tilde{v}(\tilde{v} + h)$ to be two less than that obtained using $v \bmod u$ instead of $\tilde{v}$, resulting in more efficient divisor addition. Although divisor class composition and reduction are not affected by this representation, a negative reduced basis is computed in an up adjustment, and positive in a down adjustment. By working with divisors that are already in a reduced basis, we avoid having to change basis when the corresponding type of adjustments are required.

In our implementation, we use the negative reduced basis to represent our balanced divisors. For even genus curves, no adjustments are required for typical-case inputs, so either type of reduced basis will do. However, for odd genus one up adjustment is always required for typical inputs. Having divisors always represented via a negative reduced basis ensures that base changes are not required before computing this adjustment step.

3.3 Balanced Add

Balanced Add, described in Algorithm 1, for adding divisor classes over split model curves, closely follows an optimized version of Cantor's algorithm (Algorithm 1 of [5]), with the addition of keeping v in negative reduced basis, keeping track of the balancing coefficient n, and applying adjustment steps at the end as described in [4]. The algorithm is optimized for the frequently-occurring situation where $\gcd(u_1, u_2) = 1$, based on a description due to Shanks of Gausss composition formulas for binary quadratic forms. A more efficient doubling algorithm can be obtained by specializing to the case that $D_2 = D_1$ and simplifying.

Balanced Add, and indeed all the divisor class addition algorithms presented in here, require applications of the extended Euclidean algorithm for polynomials. Throughout, we use the notation $(d, s, t) = \mathrm{XGCD}(a, b)$ to denote the output of this algorithm, specifically $d = \gcd(a, b) = as + bt$ with s, t normalized so that $\deg(s) < \deg(b) - \deg(d)$ and $\deg(t) < \deg(a) - \deg(d)$.

In the balanced setting, addition and reduction are similar to that over ramified curves, the only difference being the threshold for applying a reduction step is $\deg(u) > g + 1$ instead of $\deg(u) > g$; adjustment steps are applied when $\deg(u) = g + 1$. Reduction steps decrease the degree of the affine portion of the divisor class by at least two, so at most $\lfloor g/2 \rfloor$ steps are required to reduce the output of the composition portion to a linearly equivalent divisor whose affine part has degree at most $g + 1$.

3.4 Balanced Adjust

Balanced Adjust, described in Algorithm 2, is called after partially reducing a divisor $D = [u, v, w, n]$, for a final reduction from degree $g+1$ if necessary, and for balancing if n is outside the required range $0 \le n \le g - \deg(u)$. Balanced Adjust can be viewed as a composition of the affine portion with D_{∞^+}, when n is above the threshold (down adjustments) or D_{∞^-} when n is below (up adjustments). This can be thought of as transferring a symbolic copy of the point ∞^+ (for down) or $\infty^- = -\infty^+$ (for up) into the affine portion, keeping the divisor class the same. The number of adjustment steps required is at most $\lceil g/2 \rceil$.

4 Balanced NUCOMP

Cantor's algorithm [1] is closely related to Gauss' composition and reduction of binary quadratic forms. In 1988, Shanks [11] described an alternative to Gauss'

Algorithm 1 Balanced Add

Input: $[u_1, v_1, w_1, n_1]$, $[u_2, v_2, w_2, n_2]$, f, h, V^-.
Output: $[u, v, w, n] = [u_1, v_1, w_1, n_1] + [u_2, v_2, w_2, n_2]$.

$t_1 = v_1 + h$.
Compute $(S, a_1, b_1) = \text{XGCD}(u_1, u_2)$.
$K = a_1(v_2 - v_1) \pmod{u_2}$.
if $S \neq 1$ **then**
    Compute $(S', a_2, b_2) = \text{XGCD}(S, v_2 + t_1)$.
    $K = a_2 K + b_2 w_1$.
    if $S' \neq 1$ **then**
        $u_1 = u_1/S'$, $u_2 = u_2/S'$, $w_1 = w_1 S'$.
    $K = K \pmod{u_2}$.
    $S = S'$.
$T = u_1 K$, $u = u_1 u_2$, $v = v_1 + T$.
$w = (w_1 - K(t_1 + v))/u_2$.
$n = n_1 + n_2 + \deg(S) - \lceil g/2 \rceil$.
if $\deg(u) \leq g$ **then**
    if $\deg(v) \geq \deg(u)$ **then**
        $(q, r) = \text{DivRem}(V^- - v, u)$.
        $tv = V^- - r$, $w = w - q(v + h + tv)$, $v = tv$.
else
    while $\deg(u) > g + 1$ **do**
        if $\deg(v) = g + 1$ and $\text{lc}(v) = \text{lc}(-V^- - h)$ **then**
            $n = n + \deg(u) - g - 1$.
        else if $\deg(v) = g + 1$ and $\text{lc}(v) = \text{lc}(V^-)$ **then**
            $n = n + g + 1 - \deg(w)$.
        else $n + (\deg(u) - \deg(w))/2$.
        $u_o = u$, $u = w$.
        $(q, r) = \text{DivRem}(V^- + v + h, u)$.
        $v_t := V^- - r$, $w = u_o - q(v_t - v)$, $v = v_t$.
    $w = \text{lc}(u)w$, $u = \text{monic}(u)$.
return Balanced Adjust$([u, v, w, n], f, h, V^-)$.

Algorithm 2 Balanced Adjust

Input: $[u_a, v_a, w_a, n_a]$, f, h, V^+, where $\deg(u_a) \leq g + 1$.
Output: $[u, v, w, n] = [u_a, v_a, w_a, n_a]$, where $\deg(u) \leq g$ and $0 \leq n \leq \deg(u) - g$.

$u = u_a$, $v = v_a$, $w = w_a$, $n = n_a$,
if $n < 0$ **then**
    while $n < 0$ **do**
        $u_o = u$, $u = w$.
        $(q, r) = \mathrm{DivRem}(V^- + v + h, u)$.
        $v_t := V^- - r$, $w = u_o - q(v_t - v)$, $v = v_t$.
        $n = n + g + 1 - \deg(u)$.
    $w = \mathrm{lc}(u)w$, $u = \mathrm{monic}(u)$.
else if $n > g - \deg(u)$ **then**
    $t = -V^- - h$.
    $(q, r) = \mathrm{DivRem}(t - v, u)$.
    $v_t = t - r$, $w = w - q(v + h + v_t)$, $v = v_t$.
    while $n > g - \deg(u) + 1$ **do**
        $n = n + \deg(u) - g - 1$, $u_o = u$, $u = w$.
        $(q, r) = \mathrm{DivRem}(t + v + h, u)$.
        $v_t := t - r$, $w = u_o - q(v_t - v)$, $v = v_t$.
    if $n > g - \deg(u)$ **then**
        $n = n + \deg(u) - g - 1$, $u_o = u$, $u = w$.
        $(q, r) = \mathrm{DivRem}(V^- + v + h, u)$.
        $v_t := V^- - r$, $w = u_o - q(v_t - v)$, $v = v_t$.
    else
        $t = V^- - V^+$, $(q, r) = \mathrm{DivRem}(t, u)$.
        $v_t = v + t - r$, $w = w - q(v + v_t)$, $v = v_t$.
    $w = \mathrm{lc}(u)w$, $u = \mathrm{monic}(u)$.
return $[u, v, w, n]$.

method called NUCOMP. Instead of composing and then reducing, which results in a non-reduced intermediate quadratic form with comparatively large coefficients, the idea of NUCOMP is to start the composition process and to apply an intermediate reduction of the operands using a simple continued fraction expansion before completing the composition. The result is that the intermediate operands are smaller, and at the end the resulting quadratic form is in most cases reduced without having to apply any additional reduction steps. Jacobson and van der Poorten [6] showed how to apply the ideas of NUCOMP to divisor class group arithmetic, obtaining analogous reductions in the degrees of the intermediate polynomial operands.

Applied to our setting, the main idea is that the element $(v+y)/u \in k(C)$ is approximated by the rational function u_2/K with u_2, K from Algorithm 1. Cantor's Algorithm first computes the non-reduced divisor, and subsequently applies a reduction algorithm that can be expressed in terms of expanding the continued fraction of the quadratic irrationality $(v+y)/u$. The first several partial quotients of the simple continued fraction expansion of the rational approximation u_2/K are the same as that of $(v+y)/u$, and these can be computed without having to first compute the non-reduced divisor (u, v). Given those partial quotients, the final reduced divisor can be computed via expressions involving them and other low-degree operands, again without having to first compute the non-reduced divisor (u, v). For more details on the theory behind NUCOMP, see [8].

The most recent work on NUCOMP [5] provides further optimizations and empirical results demonstrating that it outperforms Cantor's algorithm for hyperelliptic curves of genus as small as 7, and that the relative performance improves as the genus increases. An enhanced version of NUCOMP for adding and reducing divisors without balancing, that works for curves defined over arbitrary fields and incorporates all the optimizations described in the previous section, is presented in Algorithm 3

Although this version of NUCOMP is intended for divisor class group addition on ramified model hyperelliptic curves, it also works for adding reduced affine divisors and producing a reduced output over split model curves. In [8], the authors also describe how to use this to perform arithmetic in the infrastructure of a split model hyperelliptic curve, but not in the divisor class group. It was shown that for split model curves the output of NUCOMP is always reduced for even genus curves, but that for odd genus at least one extra reduction step is required.

In the following (Algorithm 4), we present an adaptation of NUCOMP, denoted Balanced NUCOMP, for performing divisor class group arithmetic on a split model hyperelliptic curve using balanced divisor arithmetic. A more efficient doubling algorithm optimized for the case that the input divisors are equal, denoted Balanced NUDUPL, is used for our testing in Section 5 and presented as Algorithm 5 for the readers convenience. Our additions and improvements to Algorithm 3 include the following:

Algorithm 3 NUCOMP

Input: $[u_1, v_1, w_1]$, $[u_2, v_2, w_2]$, f, h.
Output: $[u, v, w]$ with $[u, v, w] = [u_1, v_1, w_1] + [u_2, v_2, w_2]$.

if $\deg(u_1) < \deg(u_2)$ **then**
    $[u_t, v_t, w_t] = [u_2, v_2, w_2]$, $[u_2, v_2, w_2] = [u_1, v_1, w_1]$.
    $[u_1, v_1, w_1] = [u_t, v_t, w_t]$.
$t_1 = v_1 + h$, $t_2 = v_2 - v_1$.
Compute $(S, a_1, b_1) = \mathrm{XGCD}(u_1, u_2)$.
$K = a_1 t_2 \pmod{u_2}$.
if $S \neq 1$ **then**
    Compute $(S', a_2, b_2) = \mathrm{XGCD}(S, v_2 + t_1)$.
    if $S' \neq 1$ **then**
        $u_1 = u_1/S'$, $u_2 = u_2/S'$. (exact divisions)
        $w_1 = w_1 S'$.
    $K = K \pmod{u_2}$.
    $S = S'$.
if $\deg(u_2) + \deg(u_1) \leq g$ **then**
    $u = u_2 u_1$, $v = v_1 + u_1 K$.
    $w = (w_1 - K(t_1 + v))/u_2$. (exact division)
    if $\deg(v) \geq \deg(u)$ **then**
        $(q, r) = \mathrm{DivRem}(v, u)$.
        $w = w + q(v + h + r)$, $v = r$.
else
    Set $r = K$, $r' = u_2$, $c' = 0$, $c = -1$, $l = -1$.
    while $\deg(r) > (\deg(u_2) - \deg(u_1) + g)/2$ **do**
        $(q, r_n) = \mathrm{DivRem}(r', r)$.
        Set $r' = r,\ r = r_n,\ c_n = c' - qc,\ c' = c,\ c = c_n,\ l = -l$.
    $t_3 = u_1 r$.
    $M_1 = (t_3 + t_2 c)/u_2$. (exact division)
    $M_2 = (r(v_2 + t_1) + w_1 c)/u_2$. (exact division)
    $u' = l(rM_1 - cM_2)$.
    $z = (t_3 + c'u')/c$. (exact division)
    $v = (z - t_1) \pmod{u'}$.
    $u = \mathrm{monic}(u')$.
    $w = (f - v(v + h))/u$.
return $[u, v, w]$.

4.1 using divisors normalized with the negative reduced basis so that in *both* even and odd genus, divisor additions generically require no further reduction nor adjustment steps after NUCOMP;
4.2 adapting NUCOMP to the balanced setting by tracking and updating the balancing coefficient n appropriately, including determining how to update the balancing coefficient n after the simple continued fraction steps;
4.3 using simple continued fraction steps of NUCOMP to eliminate an adjustment step for certain non-generic cases where the degree of the output divisor is small.

In the following subsections, we provide more details and justification for each of these modifications.

4.1 Normalization With Negative Reduced Basis

Let $[u_1, v_1, w_1, n_1]$ and $[u_2, v_2, w_2, n_2]$ be the input for Balanced NUCOMP. For split model curves, the simple continued fraction portion of NUCOMP can absorb at most one adjustment step while still ensuring that the output divisor is reduced, by setting the bound for the simple continued fraction expansion in line 22 appropriately. If the bound is set any lower than in Algorithm 4, then the resulting u polynomial ends up having degree greater than g, meaning that the divisor is not reduced.

We make the choice to normalize all our divisor class representatives using the negative reduced basis for the following reasons:

1. For odd genus, the generic case for divisor class arithmetic requires an up adjustment. Ensuring that our divisors are normalized using negative reduced basis allows us to perform this adjustment step via an extra step in the NUCOMP simple continued fraction part, so that after NUCOMP the output for the generic case is both reduced and balanced without any further steps.
2. For even genus, the generic case requires no adjustments, so either positive reduced or negative reduced basis works equally well.
3. Non-generic cases of divisor class addition over even and odd genus require either up adjustments, down adjustments or no adjustment. Over even genus, out of all cases that require adjustments, exactly half are down and half are up. Over odd genus, far more non-generic cases require an up adjustment than down. This can be seen by analyzing the computation of the balancing coefficient n in the composition portion of Algorithm 1. Line 12 states $n = n_1 + n_2 + \deg(S) - \lceil g/2 \rceil$, where the ceiling function increases the cases for which $n < 0$ for odd genus curves.

Note that it is possible to identify some non-generic cases that require a down adjustment directly from the input divisors. One could then consider using this information to change the basis to positive reduced at the beginning of the algorithm, so that the adjustment saved via NUCOMP's simple continued fraction steps is a down adjustment. However, applying the change of basis requires roughly the same amount of computation as one adjustment step in

Algorithm 4 Balanced NUCOMP

Input: $[u_1, v_1, w_1, n_1]$, $[u_2, v_2, w_2, n_2]$, f, h, V^-.
Output: $[u, v, w, n]$ with $[u, v, w, n] = [u_1, v_1, w_1, n_1] + [u_2, v_2, w_2, n_2]$.

if $\deg(u_1) < \deg(u_2)$ **then**
    $[u_t, v_t, w_t, n_t] = [u_2, v_2, w_2, n_2]$, $[u_2, v_2, w_2, n_2] = [u_1, v_1, w_1, n_1]$.
    $[u_1, v_1, w_1, n_1] = [u_t, v_t, w_t, n_t]$.
$t_1 = v_1 + h$, $t_2 = v_2 - v_1$.
Compute $(S, a_1, b_1) = \mathrm{XGCD}(u_1, u_2)$.
$K = a_1 t_2 \pmod{u_2}$.
if $S \neq 1$ **then**
    Compute $(S', a_2, b_2) = \mathrm{XGCD}(S, v_2 + t_1)$.
    $K = a_2 K + b_2 w_1$.
    if $S' \neq 1$ **then**
        $u_1 = u_1/S'$, $u_2 = u_2/S'$. (exact divisions)
        $w_1 = w_1 S'$.
    $K = K \pmod{u_2}$.
    $S = S'$.
$D = \deg(u_2) + \deg(u_1)$.
$n = n_1 + n_2 + \deg(S) - \lceil g/2 \rceil$.
if $D \leq g$ and (($n \geq 0$ and $n \leq g - D$) or $\deg(w_1) - \deg(u_2) > g$)) **then**
    $T = u_1 K$, $u = u_2 u_1$, $v = v_1 + T$.
    $w = (w_1 - K(t_1 + v))/u_2$. (exact division)
    if $\deg(v) \geq \deg(u)$ **then**
        $(q, r) = \mathrm{DivRem}(V^- - v, u)$.
        $tv = V^- - r$, $w = w - q(v + h + tv)$, $v = tv$.
else
    Set $r = K$, $r' = u_2$, $c' = 0$, $c = -1$, $l = -1$.
    while $\deg(r) \geq (\deg(u_2) - \deg(u_1) + g + 1)/2$ **do**
        $(q, r_n) = \mathrm{DivRem}(r', r)$.
        Set $r' = r$, $r = r_n$, $c_n = c' - qc$, $c = c_n$, $c' = c$, $l = -l$.
    $t_3 = u_1 r$.
    $M_1 = (t_3 + c t_2)/u_2$, $M_2 = (r(v_2 + t_1) + w_1 c)/u_2$. (exact divisions)
    $u = l(rM_1 - cM_2)$.
    $z = (t_3 + c'u)/c$. (exact division)
    $v = V^- - [(t_1 - z + V^-) \pmod{u}]$.
    $u = \mathrm{monic}(u)$.
    $w = (f - v(v + h))/u$. (exact division)
    if $\deg(z) < g + 1$ **then**
        $n = n + \deg(u_2) - \deg(r') + g + 1 - \deg(u)$.
    else
        $n = n + \deg(u_2) + \deg(r)$.
return Balanced Adjust$([u, v, w, n], f, h, V^-)$.

Algorithm 5 Balanced NUDUPL

Input: $[u_1, v_1, w_1, n_1]$, f, h, V^-.
Output: $[u, v, w, n]$ with $[u, v, w, n] = 2[u_1, v_1, w_1, n_1]$.

$t_1 = v_1 + h$, $t_2 = t_1 + v_1$.
Compute $(S, a_1, b_1) = \text{XGCD}(u_1, t_2)$.
$K = b_1 w_1$.
if $S \neq 1$ **then**
    $u_1 = u_1/S$. (exact division)
    $w_1 = w_1 S$.
$K = K \pmod{u_1}$.
$D = 2\deg(u_1)$.
$n = 2n_1 + \deg(S) - \lceil g/2 \rceil$.
if $D \leq g$ and (($n \geq 0$ and $n \leq g - D$) or $\deg(w_1) - \deg(u_1) > g$)) **then**
    $T = u_1 K$, $u = u_1^2$, $v = v_1 + T$.
    $w = (w_1 - K(t_2 + T))/u_1$. (exact division)
    if $\deg(v) \geq \deg(u)$ **then**
        $(q, r) = \text{DivRem}(V^- - v, u)$.
        $tv = V^- - r$, $w = w - q(v + h + tv)$, $v = tv$.
else
    Set $r = K$, $r' = u_1$, $c' = 0$, $c = -1$, $l = -1$.
    while $\deg(r) \geq (g + 1)/2$ **do**
        $(q, r_n) = \text{DivRem}(r', r)$.
        Set $r' = r$, $r = r_n$, $c_n = c' - qc$, $c = c_n$, $c' = c$, $l = -l$.
    $M_2 = (rt_2 + w_1 c)/u_1$. (exact division)
    $u = l(r^2 - cM_2)$.
    $z = (u_1 r + c'u)/c$. (exact division)
    $v = V^- - [(V^- - z + t_1) \pmod{u}]$.
    $u = \text{monic}(u)$.
    $w = (f - v(v + h))/u$. (exact division)
    if $\deg(z) < g + 1$ **then**
        $n = n + \deg(u_1) - \deg(r') + g + 1 - \deg(u)$.
    else
        $n = n + \deg(u_1) + \deg(r)$.
return Balanced Adjust($[u, v, w, n], f, h, V^-$).

the right direction relative to the basis. We found that in practice any savings obtained were negligible, as adjustments in the wrong direction rarely occur, so we chose not to include this functionality in our algorithm.

4.2 Adapting NUCOMP to the Balanced Setting

Most of the logic for updating the balancing coefficient n is the same as in Cantor's algorithm as presented above (Algorithm 1). The main difference is that NUCOMP does not require reduction steps, as the output is already reduced due to the simple continued fraction reduction of coefficients in lines 24–26 of Balanced NUCOMP. However, it is necessary to determine how these NUCOMP reduction steps affect the resulting balancing coefficient n.

The computation of the simple continued fraction expansion in NUCOMP implicitly keeps track of a principal divisor D_δ, such that for input divisors D_1, D_2 and the reduced output divisor D_3, $D_1 + D_2 = D_3 + D_\delta$, and knowledge of D_δ gives us the information needed to update n. Some of this is described in the version of NUCOMP from [8], but this version does not account for special cases of the last reduction step (where the leading coefficient of input v is the same as the leading coefficient of V^+ or V^-) nor the use of negative reduced basis. In our analysis we account for both, aligning with the special cases from the reduction portion of Balanced Addition (Algorithm 1) and from Balanced Adjust (Algorithm 2).

The last continued fraction step may either be a normal reduction step, a special reduction step or an adjustment step. Special reductions steps can be viewed as reductions that encounter cancellation with either ∞^+ or ∞^-. The cancellation effectively mimics a composition with ∞^+ or ∞^-, thus requiring the same accounting of the balancing coefficient n as an adjustment step. If the last step is an adjustment step, Balanced NUCOMP attempts a reduction, but a reduced basis effectively already applies composition at infinity, so the attempted reduction completes the adjustment. In both cases, the choice of either positive or negative reduced basis solely dictates the direction of the adjustment. We refer to the last simple continued fraction step as *special* if either an adjustment or special reduction step is computed; otherwise we refer to it as *normal*.

There are four possible cases for the computation of n dictated by the choice of positive or negative reduced basis and either normal or special last steps. Note that we do not include cases that arise with positive reduced basis in Algorithm 4, due to our choice of working exclusively with negative reduced basis, but we do describe the computation of n for this case below, too, as we implement and compare both versions in the next Section.

First we describe how to test for special last steps, then how the n value is computed depending on the type of basis used and whether the last step is special or normal. The last step is special exactly when $\deg(z) < g + 1$ as in line 34 of the Balanced NUCOMP algorithm, where z is given in line 30. To see this, we first recall that, as described in [8], each continued fraction step of NUCOMP corresponds to a divisor equivalent to the sum of the input divisors D_1 and D_2. Let $[u', v']$ denote the Mumford representation of the divisor corresponding to

the second-last continued fraction step. The last continued fraction step is a special step whenever $\deg(v') = g + 1$ and the leading coefficient of v' is the same as that of V^- (or V^+ if positive reduced basis is being using), because in that case cancellations in the leading coefficients of $V^- - v'$ (or $V^+ - v'$) in the computation of v cause the degree of u to be less than g, implying that the last step is special.

Comparing the computation of v in line 31 of balanced NUCOMP with the computation of v in the the reduction step of Balanced Addition (Algorithm 1), and also in any case of Balanced Adjust (Algorithm 2), we see that $v' = v_1 - z$. Thus, the conditions for the last continued fraction step being special are satisfied when $\deg(z) < g + 1$, because this implies that the degree and leading coefficients of v' and v_1 are the same. Note that $\deg(v_1) = g + 1$ and the leading coefficient of v_1 is the same as that of V^+ (or V^-) because the input divisor $[u_1, v_1]$ is given in negative (or positive) reduced basis. As stated earlier, the simple continued fraction steps of Algorithm 4 (lines 24–26) can only incorporate up to one adjustment step in addition to all the required reduction steps. Thus, a final call to Algorithm 2 is required in order to ensure that the output divisor is both reduced and balanced.

4.3 Eliminating an Adjustment for Some Non-Generic Cases

The non-balanced version of NUCOMP presented at the beginning of this section (Algorithm 3, lines 14–19) makes use of the observation that if $D = \deg(u_1/S) + \deg(u_2/S) \leq g$, then completing the composition using Cantor's algorithm will produce a divisor that is reduced without having to do any subsequent reduction steps. In the balanced setting, the corresponding balancing coefficient is $n = n_1 + n_2 + \deg(S) - \lceil g/2 \rceil$. If this divisor is not balanced, i.e. $n < 0$ or $n > g - D$, then one may apply NUCOMP's simple continued fraction-based reduction in order to compute one adjustment step, saving one of the more expensive standard adjustment steps. However, this is only beneficial if $\deg(w_1) - \deg(u_2) \leq g$, because otherwise the resulting output divisor will not be reduced due to the fact that $\deg(u)$ depends on the degree of $w_1 S/(u_2/S) = w_1/u_2$. Thus, we only finish the composition with Cantor's algorithm (lines 16–21 of Algorithm 4) if the resulting divisor is reduced and balanced, or if it is reduced and not balanced but performing a NUCOMP reduction step would result in an non-reduced divisor.

5 Empirical Results

In this section we provide empirical data to illustrate the relative performance of the composition algorithms presented above over both ramified curves and split model curves using positive and negative reduced basis representations. We implemented all the algorithms for addition and doubling in Magma as a proof of concept[4]. Therefore, the absolute timings are not of great importance.

[4] The experiments were performed on a workstation with an Intel Xeon 7550 processor that has 64 cores, each of which is 64-bit and runs at 2.00 GHz.

The reader should rather focus on the relative cost between the various algorithms and models. See `https://github.com/salindne/divisorArithmetic/tree/master/generic` for raw data, auxiliary graphs and Magma scripts of implementations used in this section.

As a preliminary benchmark, we compared addition using the versions of Cantor's algorithm described earlier and our version of NUCOMP over ramified and split model curves by computing a Fibonacci-like sequence of divisors using $D_{i+1} = D_i + D_{i-1}$, starting from two random divisors. We collected timings for all genus ranging from 2 to 50 and prime fields of sizes 2, 4, 8, 16, 32, 64, 128, 256, 512 and 1024 bits. All timings were run over random hyperelliptic curves with $h = 0$, using implementations of our algorithms that were specialized to exclude any computations with h.

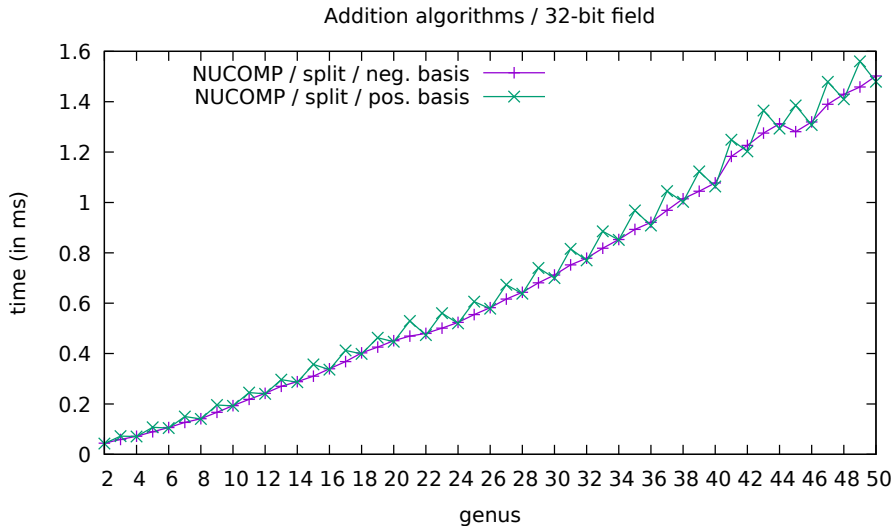
We also performed similar experiments for our doubling algorithms (Cantor's algorithm and NUDUPL, which is NUCOMP specialized to doubling a divisor) over ramified and split model curves, by computing series of thousands of additions of a divisor class with itself. The data for doubling is omitted below, as the relative performance between the various algorithms considered was the same as for addition.

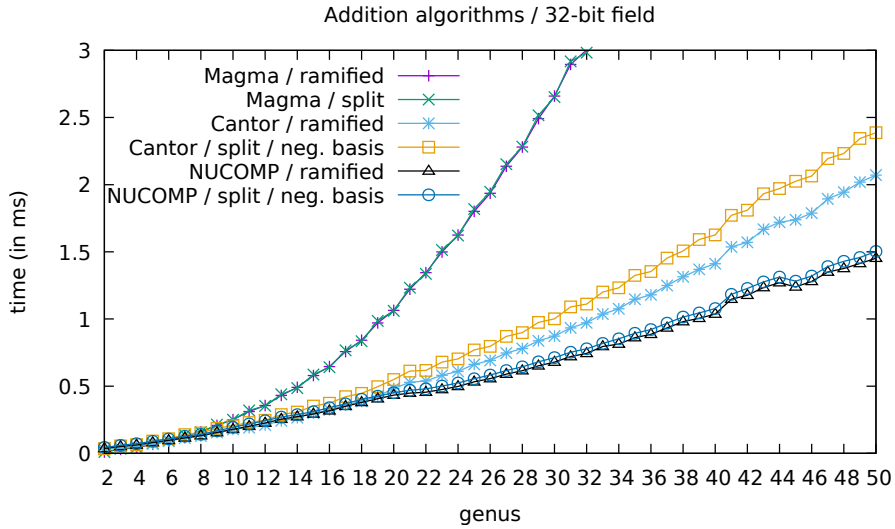
For ramified model curves, the Cantor-based algorithms we used are the same as Algorithm 1 but with the steps dealing with the balancing coefficient n removed and with divisors normalized via $v \bmod u$ as opposed to a reduced basis. We used Algorithm 3 for NUCOMP. For split model curves, the positive reduced basis algorithms are based on Algorithms 1, 2 and 4, but with divisors normalized via $V^+ - [(V^+ - v) \pmod u]$ as opposed to a negative reduced basis. We also include timings using Magma's built-in arithmetic for ramified and split model curves.

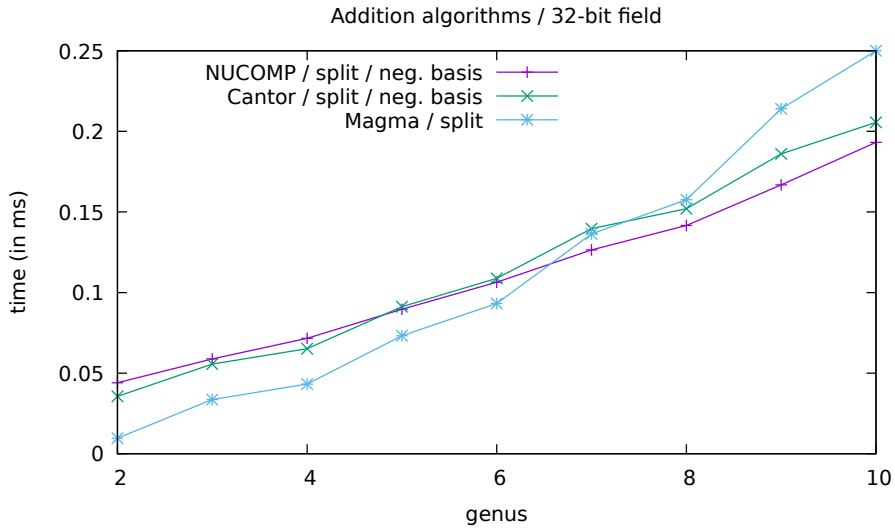
Apart from the absolute timings, we observed that the relative performances of the various algorithms do not depend on the field size. In the next figures, we illustrate our comparison for 32-bit fields only, as these results are also representative of the other field sizes. From these plots, we can draw the following conclusions:

- For split model curves, as illustrated in the first graph, negative reduced and positive reduced basis perform about the same for even genus. As expected, negative reduced basis is slightly better for odd genus due to the fact that generic cases require no adjustments steps in negative reduced basis as opposed to one adjustment step in positive reduced basis.
- The second graph shows that, for split model curves, our implementation of balanced NUCOMP rapidly becomes faster than Cantor as g grows. It also shows that, when using balanced NUCOMP, the difference between the best algorithms for split and ramified model curves is negligible for all genus. Furthermore, all of our implementations are considerably faster than Magma's built in arithmetic as the genus grows. The graph does not include timings for Magma for $g > 32$ so that the comparisons between the other algorithms are easier to see. We note that our best split model algorithm is about five times faster at genus 50.

– Not surprisingly, as shown in the last graph, for small genus ($g < 5$), Cantor's algorithms are slightly faster than the NUCOMP algorithms. Magma's built-in arithmetic is also faster for $g < 7$. We suspect that this is due to Magma's implementation having access to faster internal primitives, while our implementation has to use the generic polynomial ring setting. Even so, our implementation of NUCOMP is the fastest option for $g \geq 7$.







6 Conclusions and Future Work

Our results indicate that Balanced NUCOMP provides an improvement for computing balanced divisor class arithmetic in split model hyperelliptic curves with a cross-over as low as genus 5. As expected, our choice of normalizing v in negative reduced basis and therefore incorporating up-adjustments into NUCOMP performs equally well when compared to positive reduced over even genus, and slightly better over odd. Furthermore, our algorithm performs almost as well and sometimes better than ramified curve NUCOMP and closes the performance gap between ramified model and split model divisor arithmetic.

Integrating our algorithms directly into Magma's built in arithmetic might reduce the relative performance, either lowering or elimination any cross over points between our algorithms and Magma's arithmetic. It would be of interest to adapt NUCOMP for divisor arithmetic over non-hyperelliptic $C_{a,b}$ curves as this setting also plays a role in computational number theoretic applications [13]. Adapting NUCOMP for addition in the divisor class group of superelliptic curves based on [12] may also yield favourable results. It would also be interesting to see if explicit formulas for divisor class group arithmetic based on Balanced NUCOMP applied to arithmetic in split model hyperelliptic curves of low genus, can improve on current best [2,13].

References

1. D. Cantor, *Computing in the Jacobian of a hyperelliptic curve*, Mathematics of Computation **48** (1987), no. 177, 95–101.

2. S. Erickson, M. J. Jacobson Jr., and A. Stein, *Explicit formulas for real hyperelliptic curves of genus 2 in affine representation*, Advances in Mathematics of Communication **5** (2011), no. 4, 623–666.
3. S. D. Galbraith, *Mathematics of public key cryptography*, Cambridge University Press, 2012.
4. S. D. Galbraith, M. Harrison, and D. J. Mireles-Morales, *Efficient hyperelliptic arithmetic using balanced representation for divisors*, Algorithmic Number Theory - ANTS-VIII (Banff, Canada), Lecture Notes in Computer Science, vol. 5011, Springer-Verlag, Berlin, 2008, pp. 342–356.
5. L. Imbert and M. J. Jacobson Jr., *Empirical optimization of divisor arithmetic on hyperelliptic curves over* $\mathbb{F}_{2^m}$, Advances in Mathematics of Communication **7** (2013), no. 4, 485–502.
6. M. J. Jacobson, Jr. and A. J. van der Poorten, *Computational aspects of NUCOMP*, Algorithmic Number Theory - ANTS-V (Sydney, Australia), Lecture Notes in Computer Science, vol. 2369, Springer-Verlag, Berlin, 2002, pp. 120–133.
7. M. J. Jacobson, Jr. and H. C. Williams, *Solving the Pell equation*, CMS Books in Mathematics, Springer-Verlag, 2009, ISBN 978-0-387-84922-5.
8. M. J. Jacobson Jr., R. Scheidler, and A. Stein, *Fast arithmetic on hyperelliptic curves via continued fraction expansions*, Advances in coding theory and cryptography, World Scientific, 2007, pp. 200–243.
9. D.J.M. Mireles-Morales, *Efficient arithmetic on hyperelliptic curves with real representation*, Ph.D. thesis, University of London, 2009.
10. G.L. Mullen and D. Panario, *Handbook of finite fields*, Chapman and Hall/CRC, 2013.
11. D. Shanks, *On Gauss and composition I, II*, Proc. NATO ASI on Number Theory and Applications (R.A. Mollin, ed.), Kluwer Academic Press, 1989, pp. 163–179.
12. T. Shashka and Y. Kopeliovich, *The addition on Jacobian varieties from a geometric viewpoint*, arXiv e-prints (2019), arXiv:1907.11070v2.
13. A.V. Sutherland, *Sato-Tate Distributions*, arXiv e-prints (2016), arXiv:1604.01256.
14. A.V. Sutherland, *Fast Jacobian arithmetic for hyperelliptic curves of genus 3*, The Open Book Series **2** (2018), no. 1, 425–442.