Fourteenth International Workshop on Algebraic and Combinatorial Coding Theory
September 7–13, 2014, Svetlogorsk (Kaliningrad region), Russia *pp. 259–265*

Fast Kötter–Nielsen–Høholdt Interpolation in the Guruswami–Sudan Algorithm

Johan S. R. Nielsen jsrn@jsrn.dk
Ulm University, Department of Communications Engineering

Abstract. The Kötter–Nielsen–Høholdt algorithm is a popular way to construct the bivariate interpolation polynomial in the Guruswami–Sudan decoding algorithm for Reed–Solomon codes. In this paper, we show how one can use Divide & Conquer techniques to provide an asymptotic speed-up of the algorithm, rendering its complexity quasi-linear in n. Several of our observations can also provide a practical speed-up to the classical version of the algorithm.

1 Introduction

The computationally most demanding step of the Guruswami–Sudan algorithm [4] is finding a bivariate interpolation polynomial. Many algorithms have been proposed, both more classical with a quadratic dependence on the code length n, e.g. [6, 7], as well as approaches utilising fast multiplication methods with a resulting quasi-linear dependence on n [2, 3].

In this work we show how the Kötter–Nielsen–Høholdt algorithm[1] of [7] admits a Divide & Conquer variant to utilise fast multiplication. Our algorithm's complexity is $O(\ell^2 s^3 n) + O\tilde{}(\ell^\omega s n)$, where ℓ, s are the *list size* and *multiplicity* parameters. $O\tilde{}$ means big-O but with $\log(ns\ell)$ terms omitted, and ω is the exponent of matrix multiplication, i.e. $\omega \leq 3$.

This is not the fastest possible way to compute an interpolation polynomial, since [3] achieves $O\tilde{}(\ell^\omega s n)$, but it matches e.g. the speed of [2]. Ours is also a comparatively simple algorithm: for instance, it is trivial to apply the algorithm to Kötter–Vardy decoding [5] with varying multiplicities, while this is possible but quite complicated for the lattice-basis reduction approaches of [1–3, 6]; see [1] for a description of how to accomplish this.

The algorithm has been implemented in Sage v. 5.13 and the source code is available at http://jsrn.dk/code-for-articles.

[1] This algorithm is sometimes mistakenly attributed to Kötter only. However, it appeared first in [7], stating that it was obtained as a generalisation of an algorithm in Kötter's thesis.

2 Preliminaries and the Problem

First some notation: we will write $\mathbf{0}$ for the all-0 matrix, sub-scripted with dimensions. Likewise $\boldsymbol{I}$ is the identity matrix. For any matrix V, then $V[i,j]$ denotes the (i,j)'th entry. If V is over $\mathbb{F}[x]$ we will write $\deg V$ to denote the greatest degree among the entries of V.

For any $Q \in \mathbb{F}[x,y]$ and $w \in \mathbb{Z}_+$, denote by $\deg_w Q$ the $(1,w)$-weighted degree of Q: $\deg_w x^i y^j = i + wj$ and $\deg_w$ is then extended to polynomials by the maximal of the monomials' $\deg_w$. $\deg_w$ induces a module monomial ordering $\leq_w$, where ties are broken using the power of x.

Let $\mathbb{F}[x,y]_\ell = \{Q \in \mathbb{F}[x,y] \mid y\deg Q \leq \ell\}$; this is an $\mathbb{F}[x]$-module, and we will be working with sub-modules of it. Given a set of polynomials $\mathcal{B} \subset \mathbb{F}[x,y]_\ell$ then we denote by $\mathrm{span}(\mathcal{B})$ the $\mathbb{F}[x]$-module spanned by $\mathcal{B}$. We will be working with Gröbner bases of such modules, always on the module monomial ordering $\leq_w$. From now on, this term order is implicit when we say "Gröbner bases".

Definition 1. *For any $Q \in \mathbb{F}[x,y]$ and point $(x_0,y_0) \in \mathbb{F}^2$, then the (d_x,d_y) Hasse derivative at (x_0,y_0) for $d_x,d_y \in \mathbb{N}_0$ is the coefficient to $x^{d_x}y^{d_y}$ in $Q(x+x_0,y+y_0)$. We denote this by $\partial^{[d_x,d_y]}Q(x_0,y_0)$.*

Let $D_s = \{(d_x,d_y) \in \mathbb{N}_0^2 \mid 0 \leq d_x + d_y < s\}$. Then we say that Q has a zero of multiplicity at least s at (x_0,y_0) if $\partial^{[d_x,d_y]}Q(x_0,y_0) = 0$ for all $(d_x,d_y) \in D_s$.

We will slightly abuse notation in algorithms by using D_s as an ordered list. The order of D_s is given by $\preceq$: the lexicographical order on integer tuples, i.e. $(a_1,b_1) \preceq (a_2,b_2)$ if $a_1 < a_2$ or $a_1 = a_2 \wedge b_1 \leq b_2$.

If $Q = \sum_{i,j} q_{i,j}x^iy^j$ for $q_{i,j} \in \mathbb{F}$, then we have the formula

$$\partial^{[d_x,d_y]}Q(x_0,y_0) = \sum_{i \geq d_x}\sum_{j \geq d_y} \binom{i}{d_x}\binom{j}{d_y} q_{i,j}x_0^{i-d_x}y_0^{j-d_y}$$

Detached from the application in Guruswami–Sudan for decoding Reed–Solomon codes, the interpolation problem that we will solve is the following:

Problem 1. *Given $(x_1,y_1),\ldots,(x_n,y_n) \in \mathbb{F}^2$ with all x_i distinct, as well as $s,\ell,w \in \mathbb{Z}_+$, find a $Q \in \mathbb{F}[x,y]$ with $y\deg Q \leq \ell$ such that $\deg_w Q$ is minimal while Q has a zero with multiplicity at least s at each (x_i,y_i).*

3 The Kötter–Nielsen–Høholdt algorithm

The Kötter–Nielsen–Høholdt algorithm (KNH) for solving 1 is very short and given as Algorithm 1. We will not prove its correctness here but refer the reader to [7]. A few comments should be made before we proceed: Firstly, in [7], the initial basis is chosen as x^iy^j for all $0 \leq i < s$ and $0 \leq j \leq \ell$. It is well-known

Algorithm 1 The Kötter–Nielsen–Høholdt algorithm

Input: $\{(x_i, y_i\} \in \mathbb{F}^2, x_i$ distinct. $s, \ell, w \in \mathbb{Z}_+$
Output: Q, a solution to 1

```
1 B ← {1, y, ..., y^ℓ}
2 for i = 1, ..., n do
3     for (d_x, d_y) ∈ D_s do
4         b_t ← arg min_{b_j ∈ B}{deg_w b_j if ∂^[d_x,d_y] b_j(x_i, y_i) ≠ 0 else ∞}
5         B ← {b_j − (∂^[d_x,d_y] b_j(x_i, y_i)/∂^[d_x,d_y] b_t(x_i, y_i)) b_t | b_j ∈ B \ {b_t}}
6                 ∪{(x − x_i) b_t}
7 return arg min_{b_j ∈ B}{deg_w b_j}
```

that one can simply choose y^j for $0 \leq j \leq \ell$, without changing the correctness of the algorithm.

Secondly, the proof of correctness actually establishes that after the i'th iteration of the outer loop, $\mathcal{B}$ is a Gröbner basis of $M_1 \cap \ldots \cap M_i$, where $M_j \subset \mathbb{F}[x, y]_\ell$ consists of all polynomials with a zero of multiplicity at least s at (x_j, y_j). Furthermore, each iteration of the inner loop further refines $\mathcal{B}$ to be a Gröbner basis of those polynomials Q which also have $\partial^{[d_x,d_y]} Q(x_i, y_i) = 0$.

Proposition 1. *The complexity of the KNH is* $O(\ell^2 s^3 n^2)$.

Proof. Firstly, any $b \in \mathcal{B}$ has $x\deg b < sn$ since for each point the same index t can be chosen in Line 4 at most s times. That is because if for some d_x then $\partial^{[d_x-1,d_y]} b(x_i, y_i) = 0$ for all d_y, then $\partial^{[d_x,d_y]}((x - x_i)b(x_i, y_i)) = 0$ for all d_y; i.e. if some b_t is chosen, it will be replaced with $(x - x_i)b_t$ so it will not be chosen again for the same d_x. Thus the maximal x-degree in $\mathcal{B}$ increases at most s for every iteration of the outer loop.

Now in the $O(s^2 n)$ iterations of the inner loop, we compute $\ell + 1$ Hasse derivatives of basis elements, as well as ℓ linear combinations of two basis elements. By the $x$deg on basis elements, either such operation costs $O(\ell sn)$. □

4 Manipulating Hasse Derivatives

We will start with some observations on the computation and manipulation of the Hasse derivatives during the inner loop of the algorithm. For any $Q = \sum_{i=0}^{\ell} Q_i y^i \in \mathbb{F}[x, y]_\ell$ and $p \in \mathbb{F}[x]$, we will denote by $Q \bmod p$ the polynomial $\sum_{i=0}^{\ell} (Q_i \bmod p) y^i$. Likewise, for a set $\mathcal{B}$ of $\mathbb{F}[x, y]$-elements, we will denote by $\mathcal{B} \bmod p$ the set $\{b \bmod p \mid b \in \mathcal{B}\}$.

Lemma 1. *For any $Q \in \mathbb{F}[x,y]$, point $(x_0,y_0) \in \mathbb{F}^2$, and $(d_x,d_y) \in D_s$, then*

$$\partial^{[d_x,d_y]}Q(x_0,y_0) = \partial^{[d_x,d_y]}(Q \bmod (x-x_0)^s)(x_0,y_0)$$

Proof. Let $Q = \sum_{i=0}^{\ell} Q_i y^i$ and $\hat{Q} = \sum_{i=0}^{\ell} \hat{Q}_i y^i = Q \bmod (x-x_0)^s$. Then there exist $q_0,\ldots,q_\ell \in \mathbb{F}[x]$ such that $Q_i = \hat{Q}_i + q_i(x-x_0)^s$ for every i. But then $Q_i(x+x_0) = \hat{Q}_i(x+x_0) + x^s q_i(x+x_0)$. The lemma now follows from the definition of $\partial^{[d_x,d_y]}$. □

We need the above lemma for our Fast KNH, but together with another observation it can even be used in the original KNH to speed up calculations: for each point, we can compute all the Hasse derivatives for each basis element just once and then update them during the iterations of the inner loop. That is possible since Hasse derivatives change straightforwardly under the operations performed: first, represent the derivatives of a given element b as an upper anti-triangular matrix $H = [\partial^{[d_x,d_y]}b(x_i,y_i)]_{(d_x,d_y)\in D_s} \in \mathbb{F}^{s\times s}$. Then the linear combinations in Line 6 of Algorithm 1 can simply be reflected as linear combinations of these Hasse matrices. Furthermore, if H_t is the Hasse matrix for b_t, then the one for $(x-x_i)b_t$ is simply H_t shifted down by one row, and the new first row set to all-zero; the elements now outside the upper anti-diagonal can be set to zero or ignored.

In the original KNH, this can reduce the total cost of computing with Hasse derivatives to $O(\ell s^4 n + sn^2) + \tilde{O}(\ell s n)$; due to space limitations we omit the details on this. Since the cost of updating $\mathcal{B}$ in the inner loop still incurs cost $O(\ell^2 s^3 n^2)$, the overall complexity remains unchanged. However, I can remark that the above optimisation drastically sped up my own software implementation of the KNH algorithm.

This bottleneck of updating $\mathcal{B}$ is exactly what is handled in the Fast KNH, described in the next section, allowing us to end up with a complete algorithm which is quasi-linear in n.

5 Fast KNH: A Divide & Conquer Variant

The main idea of the Fast KNH is to completely avoid working with the unreduced $\mathcal{B}$ in the inner loop, and only work with $\mathcal{B} \bmod (x-x_i)^s$. The operations to perform only depend on the Hasse matrices, so we do not actually manipulate $\mathcal{B}$ in the inner loop; instead the operations are "recorded" as a matrix $T \in \mathbb{F}[x]^{(\ell+1)\times(\ell+1)}$, and when continuing the interpolation with the next point, they are applied to $\mathcal{B}$ as $T(\mathcal{B} \bmod (x-x_{i+1})^s)$. In particular, the operations of

one iteration of the inner loop can be represented as the matrix U:

$$U = \boldsymbol{I}_{(\ell+1)\times(\ell+1)} - \left[\mathbf{0}_{(\ell+1)\times(t-1)} \mid \boldsymbol{u}_t^\top \mid \mathbf{0}_{(\ell+1)\times(\ell-t+2)}\right] \tag{1}$$

$$\boldsymbol{u}_t = (H_1[d_x,d_y]/H_t[d_x,d_y], \ldots, H_{t-1}[d_x,d_y]/H_t[d_x,d_y],$$
$$1-(x-x_i),\ H_{t+1}[d_x,d_y]/H_t[d_x,d_y], \ldots, H_{\ell+1}[d_x,d_y]/H_t[d_x,d_y])$$

The list of points (x_i, y_i) to process is then structured into a binary tree to minimise the representation of $\mathcal{B}$ necessary at any given time. This results in two sub-algorithms: InterpolatePoint as well as InterpolateTree. InterpolateTree is the main entry point, called with the basis $\mathcal{B} = \{1, y, \ldots, y^\ell\}$.

Proposition 2. InterpolatePoint *is correct. It has computational complexity* $O(\ell^2 s^3)$.

Proof sketch. Only how the degrees δ_j are updated has not already been discussed. For a given iteration of the loop, let b_j refer to the elements of $T\mathcal{B}$ where T is as in the beginning of the iteration, while b'_j are the elements of $T'\mathcal{B}$ where T' is T at the end of the iteration. Let also B be the set of b_j such that $\partial^{[d_x,d_y]} b_j(x_i, y_i) \neq 0$, and let B' be the set of b'_j for the same indices. Clearly $\deg_w b'_t = \deg_w b_t + 1$, so the update in Line 8 is correct. We claim $\deg_w b'_j = \deg_w b_j$ for $j \neq t$: by the choice of b_t then $\deg_w b_t \leq \deg_w b_j$, which means that $\deg_w b'_j \leq \deg_w b_j$. If $\deg_w b_t < \deg_w b_j$ then clearly $\deg_w b'_j = \deg_w b_j$. Otherwise, assume that $\deg_w b_t = \deg_w b_j$. Now, $T\mathcal{B}$ is a Gröbner basis of $\mathrm{span}(T\mathcal{B})$ (recall that this was part of the proof of the original KNH in [7]), so likewise B is a Gröbner basis of $\mathrm{span}(B)$. Since $\mathrm{span}(B') \subset \mathrm{span}(B)$ then $b'_j \in \mathrm{span}(B)$. But then $\deg_w b'_j$ cannot be less than the $\deg_w$ of all the elements in B.

Due to lack of space, we omit the details on the computational complexity. □

Proposition 3. InterpolateTree *is correct. It has computational complexity* $O(\ell^2 s^3 n) + O\tilde{}(\ell^\omega s n)$*, where* n *is the number of input points.*

Proof. Correctness follows inductively by the correctness of InterpolatePoint, since $\hat{\mathcal{B}}_1 = \hat{\mathcal{B}} \mod \prod_{h=i_1}^{t}(x-x_i)^s = \mathcal{B} \mod \prod_{h=i_1}^{t}(x-x_i)^s$.

Let $C(n)$ denote the complexity on n input points, ignoring the costs of calls to InterpolatePoint. Since $\deg T_1, \deg T_2 \leq sn/2$ then $C(n) = 2C(n/2) + O\tilde{}(\ell^\omega sn/2)$, which means $C(n) \in O\tilde{}(\ell^\omega sn)$. Adding the cost of n calls to InterpolatePoint yields the result. Computing the $O(2n)$ moduli polynomials has negligible cost $O\tilde{}(sn)$. □

References

[1] M. Alekhnovich. Linear Diophantine Equations Over Polynomials and Soft Decoding of Reed–Solomon Codes. *IEEE Trans. Inf. Theory*, 51(7):2257–2265, July 2005.

Algorithm 2 InterpolatePoint

Input: $(x_i, y_i) \in \mathbb{F}^2$, $s, \ell, w \in \mathbb{Z}_+$, $\hat{\mathcal{B}}$, and $\{\delta_j\}_j$. Here $\hat{\mathcal{B}} = \mathcal{B} \mod (x - x_i)^s$ where $\mathcal{B} \subset \mathbb{F}[x, y]_\ell$ is a Gröbner basis of $\mathrm{span}(\mathcal{B})$, and $\delta_j = \deg_w b_j$ for each $b_j \in \mathcal{B}$.

Output: $T \in \mathbb{F}[x]^{(\ell+1)\times(\ell+1)}, \{\hat{\delta}_j\}_j$. Here $T\mathcal{B}$ is a Gröbner basis of $\mathrm{span}(\mathcal{B}) \cap M_i$, and $\hat{\delta}_j = \deg_w \hat{b}_j$ for each $\hat{b}_j \in T\mathcal{B}$.

$H_j = [\partial^{[d_x,d_y]} b_j(x_i, y_i)]_{(d_x,d_y)\in D_s}$ for each $\hat{b}_j \in \hat{\mathcal{B}}$
$T = \boldsymbol{I}_{(\ell+1)\times(\ell+1)}$
for $(d_x, d_y) \in D_s$ **do**
  $t \leftarrow \arg\min_{t\in\{1,\ldots,\ell+1\}}\{\delta_j \text{ if } H_j[d_x, d_y] \neq 0 \text{ else } \infty\}$
  $H_j = H_j - (H_j[d_x, d_y]/H_t[d_x, d_y])H_t$, for $j \neq t$
  $H_t = [\ \mathbf{0}_{(\ell+1)\times 1} \mid \dot{H}_t^\top\ ]^\top$ where $\dot{H}_t$ is H_t with the last row removed
  $T = UT$, where U is as in (1)
  $\delta_t = \delta_t + 1$
return $T, \{\delta_j\}_j$

Algorithm 3 InterpolateTree

Input: $(x_{i_1}, y_{i_1}\}, \ldots, (x_{i_2}, y_{i_2}) \in \mathbb{F}^2$, $s, \ell, w \in \mathbb{Z}_+$, $\hat{\mathcal{B}}$ and $\{\delta_j\}_j$. Here $\hat{\mathcal{B}} = \mathcal{B} \mod \prod_{h=i_1}^{i_2}(x - x_h)^s$ where $\mathcal{B} \subset \mathbb{F}[x, y]_\ell$ is a Gröbner basis of $\mathrm{span}(\mathcal{B})$, and $\delta_j = \deg_w b_j$ for each $b_j \in \mathcal{B}$.

Output: $T \in \mathbb{F}[x]^{(\ell+1)\times(\ell+1)}, \{\hat{\delta}_j\}_j$. Here, $T\mathcal{B}$ is a Gröbner basis of $\mathrm{span}(\mathcal{B}) \cap M_{i_1} \cap \ldots \cap M_{i_2}$ and $\hat{\delta}_j = \deg_w \hat{b}_j$ for each $\hat{b}_j \in T\mathcal{B}$.

if $i_1 = i_2$ **then return** InterpolatePoint$((x_{i_1}, y_{i_1}),\ \hat{\mathcal{B}},\ \{\delta_j\}_j)$
else
  $t \leftarrow \lfloor (i_1 + i_2)/2 \rfloor$; $\hat{\mathcal{B}}_1 \leftarrow \hat{\mathcal{B}} \mod \prod_{h=i_1}^{t}(x - x_h)^s$
  $(T_1, \{\delta_j\}) \leftarrow$ InterpolateTree$((x_{i_1}, y_{i_1}), \ldots, (x_t, y_t),\ \hat{\mathcal{B}}_1,\ \{\delta_j\}_j)$
  $\hat{\mathcal{B}}_2 \leftarrow T_1\hat{\mathcal{B}} \mod \prod_{h=t+1}^{i_2}(x - x_h)^s$
  $(T_2, \{\delta_j\}) \leftarrow$ InterpolateTree$((x_{t+1}, y_{t+1}), \ldots, (x_{i_2}, y_{i_2}),\ \hat{\mathcal{B}}_2,\ \{\delta_j\}_j)$
  return $(T_2T_1, \{\delta_j\})$

[2] P. Beelen and K. Brander. Key equations for list decoding of Reed–Solomon codes and how to solve them. *J. Symb. Comp.*, 45(7):773–786, 2010.

[3] H. Cohn and N. Heninger. Ideal forms of Coppersmith's theorem and Guruswami–Sudan list decoding. *arXiv*, 1008.1284, 2010.

[4] V. Guruswami and M. Sudan. Improved Decoding of Reed–Solomon Codes and Algebraic Geometry Codes. *IEEE Trans. Inf. Theory*, 45(6):1757–1767, 1999.

[5] R. Kötter and A. Vardy. Algebraic Soft-Decision Decoding of Reed-Solomon Codes. *IEEE Trans. Inf. Theory*, 49(11):2809–2825, 2003.

[6] K. Lee and M. E. O'Sullivan. List Decoding of Reed–Solomon Codes from a Gröbner Basis Perspective. *J. Symb. Comp.*, 43(9):645 – 658, 2008.

[7] R. R. Nielsen and T. Høholdt. Decoding Reed–Solomon codes beyond half the minimum distance. In *Coding Theory, Cryptography and Related Areas*, page 221–236. Springer, 1998.