ALGORITHMS FOR THE MANUSCRIPT "HAMMING AND GILBERT BOUNDS FOR POLY-ALPHABETIC CODES WITH THE L1 DISTANCE"

E. J. GARCÍA-CLARO AND ISMAEL GUTIERREZ ‡

Abstract. This document presents several SageMath implementations useful to apply the main results of the manuscript entitled "Hamming and Gilbert bounds for poly-alphabetic codes with the l_1 distance".

Algorithms to compute $\eta_r(\mathcal{X})$ and $\gamma_r(\mathcal{X})$ and $\overline{B}_r$

Algorithm 1 provides the code in *SageMath* to calculate $\eta_r(\mathcal{X})$.

To use the *Python* functions given in algorithms in this note, one can import them into a SageMath worksheet by writing `load("GitHub_L1Codes.py")`[1] at the beginning of the worksheet.

Data: `r` $\in \mathbb{Z}_{\geq 0}$ and the *Python* list `m` $= [m_1, ..., m_n]$ where $\mathcal{X} = \prod_{i=1}^{n} [0, m_i - 1]$.
Result: `eta(r, m)` computes $\eta_r(\mathcal{X})$.

```
def eta(r, m):
    if r==0:
        return 1

    n=len(m); indices = range(1, n+1); eta = 0

    for delta in range(r + 1):
        for J in Subsets(indices):
            sum_m_J = sum(m[i-1] for i in J)
            if delta - sum_m_J >= 0:
                binom_term = binomial(n + delta - sum_m_J - 1
                           , delta - sum_m_J)
                eta += (-1) ** len(J) * binom_term

    return eta
```

Algorithm 1: A *Python* function that applies Theorem 3.8 to compute $\eta_r(\mathcal{X})$ in *SageMath*.

Let $x \in Inm(\mathcal{X})$, $n \in \mathbb{Z}_{\geq 1}$, $\delta \in \mathbb{Z}_{>0}$, p_δ be the coefficient of degree δ of $p(t, x)$. The function in Algorithm 4 uses the functions given in Algorithms 2, 3 to provide the *SageMath* code to calculate

---

Date: Apr 2025.
[1]The document `"GitHub_L1Codes.py"` can be donwloaded from https://github.com/egarcia-claro/L1Codes

p_δ for $\mathcal{X} = \prod_{i=1}^{n}[0, m_i - 1]$ when $E = \{i \in [n] : 2 \mid m_i\}$ is arbitrary.

Data: `delta` $\in \mathbb{Z}_{\geq 0}$ and the *Python* list `m` $= [m_1, ..., m_n]$ where $\mathcal{X} = \prod_{i=1}^{n}[0, m_i - 1]$.
Result: `p_delta_o(delta, m)` computes the coefficient p_δ of degree δ of $p(t, \text{any innermost point})$ when $E = \{i \in [n] : 2 \mid m_i\} = \emptyset$.

```
def p_delta_o(delta, m):
    if delta==0:
        return 1

    n=len(m); l = [ceil((mi - 1) / 2) for mi in m]
    indices=range(1, n+1)
    E = {i for i in indices if m[i-1] % 2 == 0}
    partial=sum(li for li in l)
    p_delta = 0

    if len(E) != 0:
        raise ValueError("The m_i’s must be odd")

    elif delta == partial:
        return 2**n

    P_n= Subsets(indices)
    for J in P_n:
        if len(J) > 0:
            for A in Subsets(J):
                sum_l_A = sum(l[i-1] + 1 for i in A)
                if delta - sum_l_A >= 0:
                    binom_term = binomial(len(J) + delta - sum_l_A
                                 - 1, delta - sum_l_A)
                    p_delta += (-1)**(n - len(J) + len(A))
                             * 2**len(J) * binom_term

    return p_delta
```

Algorithm 2: An algorithm that applies Theorem 3.10 (part 1) to compute p_δ in *SageMath*.

Data: `delta` $\in \mathbb{Z}_{\geq 0}$ and the *Python* list `m` $= [m_1, ..., m_n]$ where $\mathcal{X} = \prod_{i=1}^{n}[0, m_i - 1]$.
Result: `p_delta_o(delta, m)` computes the coefficient p_δ of degree δ of $p(t,$ any innermost point) when $E = \{i \in [n] : 2 \mid m_i\} = [n]$.

```
def p_delta_e(delta, m):
    if delta==0:
        return 1

    n=len(m); l = [ceil((mi - 1) / 2) for mi in m]
    indices=range(1, n+1)
    E = {i for i in indices if m[i-1] % 2 == 0}
    partial=sum(li for li in l)
    p_delta = 0

    if len(E) != n:
        raise ValueError("The m_i's must be even")

    if delta == partial:
        return 1

    P_n = Subsets(indices)
    for J in P_n:
        if len(J) > 0:  # Ensure J is non-empty
            J_c = set(indices) - set(J)
            u_J_delta = (-1)**(len(J)) if delta ==
                        sum(l[i-1] for i in J_c) else 0

            term_sum = u_J_delta

            for A in Subsets(J):
                if len(A) > 0:
                    for B in Subsets(A):
                        sum_l_B_J_c = sum(l[i-1] for i in B)
                                    + sum(l[i-1] for i in J_c)
                        if delta - sum_l_B_J_c >= 0:
                            binom_term = binomial(len(A) + delta
                                         - sum_l_B_J_c - 1, delta
                                         - sum_l_B_J_c)
                            term_sum += (-1)**(len(J) - len(A)
                                        + len(B))  *  2**len(A)
                                        *  binom_term

            p_delta += term_sum

    return p_delta
```

Algorithm 3: An algorithm that applies Theorem 3.10 (part 2) to compute p_δ in *SageMath*.

Data: `delta` $\in \mathbb{Z}_{\geq 0}$ and the *Python* list `m` $= [m_1, ..., m_n]$ where $\mathcal{X} = \prod_{i=1}^{n}[0, m_i - 1]$.
Result: `p_delta(delta, m)` computes the coefficient p_δ of degree δ of $p(t, \text{any innermost point})$ when $E = \{i \in [n] : 2 \mid m_i\}$ is arbitrary.

```
def p_delta(delta, m):
    if delta==0:
        return 1

    n=len(m); indices=range(1, n+1)
    m_e=[m[i-1] for i in indices if m[i-1] % 2 == 0]
    partial_e=sum(ceil((m - 1) / 2) for m in m_e)
    m_o=[m[i-1] for i in indices if m[i-1] % 2 != 0]
    p_delta = 0

    return sum(p_delta_e(j, m_e) *  p_delta_o(delta-j, m_o)
               for j in range(partial_e + 1))
```

Algorithm 4: An algorithm that applies Theorem 3.10 (part 3) to compute p_δ in *SageMath*. It works even when $E = \emptyset$ or $E = [n]$, because `p_delta_e(0, [])=p_delta_o(0, [])=1`.

Data: `r` $\in \mathbb{Z}_{\geq 0}$ and the *Python* list `m` $= [m_1, ..., m_n]$ where $\mathcal{X} = \prod_{i=1}^{n}[0, m_i - 1]$.
Result: `gamma(r, m)` computes $\gamma_r(\mathcal{X})$.

```
def gamma(r, m):
    if r==0:
        return 1

    return 1 + sum(p_delta(delta, m) for delta in range(1, r+1))
```

Algorithm 5: Since $\gamma_r(\mathcal{X}) = p(1, x)_{\leq r} = 1 + \sum_{\delta=1}^{r} p_\delta$ (by Lemma 3.2, part 2, and Theorem 3.6) and $p_\delta =$ `p_delta(delta, m)` (where `p_delta(delta, m)` is given as in Algorithm 4), this function computes $\gamma_r(\mathcal{X})$.

Algorithms 7 and 10 compute the polynomials $P(t, a)$ and $\overline{p}(t)$ described in Lemma 3.2, parts 1 and 2, respectively.

Data: $j, m \in \mathbb{Z}_{\geq 0}$ and $x \in \mathcal{X}$.
Result: `sphere_size(j,x,m)` computes $|S_j(x)|$ computes in $[0, m-1]$.

```
def sphere_size(j, x, m):
    l_min = max(x, m - (x + 1)); l_max = min(x, m - (x + 1))

    if j == 0:
        return 1
    elif 0 < j <= l_max:
        return 2
    elif l_max < j <= l_min:
        return 1
    else:
        return 0
```

Algorithm 6: An algorithm that applies Lemma 3.5 to compute the size of an j-sphere in $[0, m-1]$ in *SageMath*.

Data: the *Python* list $\mathtt{m} = [m_1, ..., m_n]$ where $\mathcal{X} = \prod_{i=1}^{n}[0, m_i - 1]$, and $a \in \mathcal{X}$.
Result: `compute_p_t(t,a,m)` computes the local distance enumerator polynomial $p(t, a)$.

```
def compute_p_t(t,a, m):
    product = 1

    for i in range(len(a)):
        l_i = max(a[i], m[i] - (a[i] + 1))
        sum_expr = sum(sphere_size(j, a[i], m[i]) * t**j for j in range(l_i + 1))
        product *= sum_expr
    return product
```

Algorithm 7: An algorithm that applies Lemma 3.2 (part 1) to compute the local distance enumerator polynomial $p(t, a)$ in *SageMath*.

Data: $n \in \mathbb{Z}_{>0}$, $m \in \mathbb{Z}_{>1}$
Result: `generate_representative_orbits(m,n)` computes a set of representatives $\mathfrak{T}$ of orbits under the action of $G = C_2^n \rtimes S_n$ on $\mathcal{X} = [0, m-1]^n$.

```
def generate_representative_orbits(m,n):
    from itertools import product

    bounds = list(range(0, (m - 1) // 2 + 1)); reps = [ ]
    cartesian_product=list(product(bounds, repeat=n))

    for candidate in cartesian_product:
        if list(candidate) == sorted(candidate):
            reps.append(candidate)
    return reps
```

Algorithm 8: An algorithm that applies Lemma 3.1 (part 2) to compute a set of representative of orbit in *SageMath*.

Data: $m \in \mathbb{Z}_{\geq 0}$ and $a \in \mathfrak{T}$, where $\mathfrak{T}$ is a set of representatives of orbits under the action of $G = C_2^n \rtimes S_n$ on $\mathcal{X} = [0, m-1]^n$.
Result: `compute_stabilizer_size(a, m)` computes $|Stab_G(a)|$.

```
def compute_stabilizer_size(a, m):
    from collections import Counter
    from math import factorial

    n = len(a);  I_a = set()

    for i in range(n):
        reflected_value = m - (a[i] + 1)
        if reflected_value in a:
            I_a.add(i)

    multiplicities = Counter(a)
    product_factorial = 1
    for count in multiplicities.values():
        product_factorial *= factorial(count)

    stabilizer_size = (2 ** len(I_a)) * product_factorial
    return stabilizer_size
```

Algorithm 9: An algorithm that applies Lemma 3.1 (part 3) to compute the size of the stabilizer of an element (in $\mathfrak{T}$) in *SageMath*.

Data: t is the variable in $R =$ `PolynomialRing(QQ, t)`, $m \in \mathbb{Z}_{>1}$, $n \in \mathbb{Z}_{>0}$
Result: `compute_p_bar(t, m, n)` computes the polynomial $\overline{p}(t) \in \mathbb{Q}[t]$ of $\mathcal{X} = [0, m-1]^n$.

```
def compute_p_bar(t, m, n):
    from math import factorial

    T = generate_representative_orbits(m,n); total_sum = 0
    for a in T:
        stab_size = compute_stabilizer_size(a, m)
        orbit_size = QQ(factorial(n) * (2**n)) / stab_size
        p_t_a = compute_p_t(t, a, [m]*n)
        total_sum += orbit_size * p_t_a

    return total_sum / m**n
```

Algorithm 10: An algorithm that applies Lemma 3.2 (part 2) to compute $\overline{p}(t)$, this is a generating function for the average size of an sphere (in $\mathcal{X} = [0, m-1]^n$) in *SageMath*.

Departamento de Matemáticas, Universidad Autónoma Metropolitana, Unidad Iztapalapa, Código postal 09340, Ciudad de México - México
Email address: `eliasjaviergarcia@gmail.com`

Department of Mathematics and Statistics, Universidad del Norte, Km 5 via a Puerto Colombia, Barranquilla - Colombia
Email address: `isgutier@uninorte.edu.co`