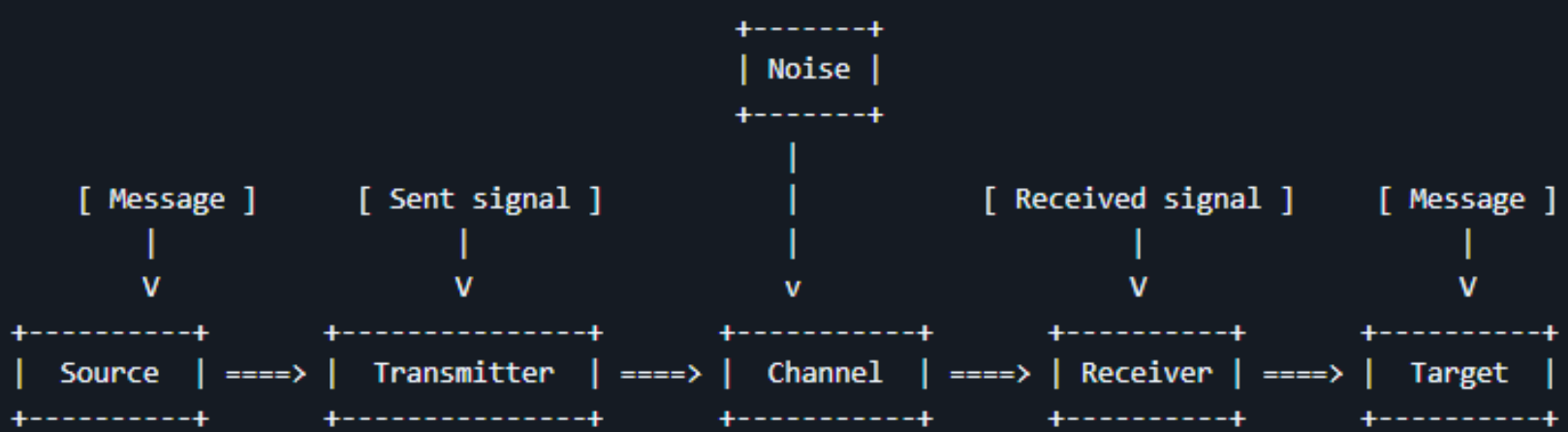
Linear Error-Correcting Codes: Disjoint Spheres Theorem

Cristina Dueñas Navarro

Overview:

This project formalizes the concept of Linear Error-Correcting Codes (ECC) and minimum distance, and verifies that the minimum distance of a code equals the minimum of the distances between different codewords of the code, in order to verify the Disjoint Spheres Theorem, which states that spheres of a small enough radius centered on different codewords of a code are disjoint. This theorem implies that nearest neighbour decoding uniquely and correctly decodes any received vector `y = c + e`, where `e` is a 'small enough error' to the sent codeword `c`.

The motivation behind ECC is that sending a message through a communication channel can introduce errors during signal transmission:



For a source on this topic, you can consult the book 'Fundamentals of Error-Correcting Codes' by W. Cary Huffman and Vera Pless, 2003.

- Global variables `F` finite field with cardinality `q` : I defined these as global variables since it's the basis of all the constructions I use in this project. This might not be convenient when expanding ECC theory formalizations, but that's easy to solve.

```
7       -- For this project, I'm globally defining the field `F` = F_q of `q` elements
8       variable (F : Type*) [Field F] [Fintype F] [DecidableEq F]
9       variable {q : ℕ} (hq : Fintype.card F = q)
```

```
/- Some relevant definitions and theorems:
-- Subspace `k` of a module `M`
Subspace k M

-- **Hamming weight or norm** (number of non-zero entries in the vector `x`)
hammingNorm x

-- **Hamming distance** (number of positions where x and y differ)
hammingDist x y

-- Basic properties of hammingDist:
  -- Non-negativity
  hammingDist_nonneg

  -- Non-degeneracy
  hammingDist_eq_zero

  -- Symmetry
  hammingDist_comm

  -- Triangle inequality
  hammingDist_triangle

-- Relations between hammingDist and hammingNorm:
  -- hammingDist x 0 = hammingNorm x
  hammingDist_zero_right

  -- hammingDist x y = hammingNorm (-x + y)
  hammingDist_eq_hammingNorm
-/
```

- `LinearCode` : This is the definition of linear ECC. I defined it as a structure so it contains the space itself (the code) and its dimension. The dimension is not needed for this project.

```
42      -- An **[n, k]_q-linear code** is a vector subspace `C` of `F^n_q` of dimension `k`
43      structure LinearCode (n k : ℕ) where
44        -- The scalars are in `F_q`, the vectors are in `(Fin n → F) = F_q^n`
45        space : Subspace F (Fin n → F)
46        dim : Module.finrank F space = k
47      -- The elements of `C` are called **codewords**.
```

- `minDist` : The minimum distance of a code. I defined it as non-computable and as an infimum because that was enough for this project and easier for the proofs. It's trivial to see that it's a minimum, as the fields are finite. As for the computability, a fun fact is that it is actually an NP-hard problem in general (that's why some specific families of codes such as Reed-Solomon are typically used). For more information, see the paper 'The intractability of computing the minimum distance of a code', Alexander Vardy, IEEE Transactions on Information Theory, 43(6):1757–1766, Nov 1997.

```
49      -- The **minimum distance** `d(C)` of a code `C` is the minimum weight of its non-zero elements
50      -- (Note that this is defined here as an infimum, not a minimum.
51      -- It should be easy to prove that the infimum is actually a minimum,
52      -- as `C` itself is a finite set,
53      -- but we won't need that fact in this project.)
54      noncomputable def minDist {n k : ℕ} (C : LinearCode F n k) : ℕ :=
55        sInf {w | ∃ x : Fin n → F, x ∈ C.space ∧ x ≠ 0 ∧ w = hammingNorm x}
```

- `LinearCodeWithDist` : This is a `LinearCode` with a lower bound `d` for the minimum distance. Hence, it is defined as a structure with those two elements.

```
57      -- An **[n, k, d]-linear code** is an [n, k]-linear code with distance bounded from below by `d`
58      structure LinearCodeWithDist (n k d : ℕ) where
59        code : LinearCode F n k
60        dist_lower_bound : d ≤ minDist F code
```

- Lemma `minDist_eq_min_pairwise_dist` : The minimum distance of a code coincides with the minimum of the distances of different codewords. Actually, we prove the theorem for the infimums. As previously stated, it is easy to see that for finite fields, the infimums are minimums.

```
62      --Linearity implies the following lemma:
63      -- **Lemma:** The minimum distance of a code coincides with the
64      -- minimum of the distances of different codewords
65      -- (Actually, we will prove the theorem for the infimums. It is easy to see that for finite fields,
66      -- the infimums are minimums)
67      -- (After proving this theorem, lean would show a warning that the `Fintype F` assumption is not
68      -- needed, so I am omitting it for this proof)
69      omit [Fintype F] in
70      theorem minDist_eq_min_pairwise_dist {n k : ℕ} (C : LinearCode F n k) :
71          minDist F C = sInf {d | ∃ x ∈ C.space, ∃ y ∈ C.space, x ≠ y ∧ d = hammingDist x y} := by
```

- Lemma `minDist_eq_min_pairwise_dist` : We prove that both infimums are equal by proving that the sets are equal (`congr 1`) since each is contained in the other (`ext`):

```
72        -- Unfold the definition of `minDist` so we are looking at sInf(Set W) = sInf(Set D)
73        unfold minDist
74        -- Convert `hammingNorm x` to `hammingDist x 0`
75        simp_rw[← hammingDist_zero_right]
76        -- Goal moves to proving the sets W and D are equal
77        congr 1
78        -- Goal moves to proving that `element` belongs to W if, and only if, it belongs to D
79        -- (In set theory, the axiom of extensionality states that two sets are equal if and only if they
80        -- contain the same elements)
81        ext element
82        -- Now we use that `(x ∈ {y | p y}) = p x`, so the goal moves to proving that `element` satisfies
83        -- the properties defining W if, and only if, it satisfies the properties defining D
84        simp only [Set.mem_setOf_eq]
85        -- Split the "if and only if" into two implications, and prove them separately
86        constructor
```

- For proving that the weight set is contained in the distance set, we use that w(x) = d(x, 0) (`hammingDist_zero_right`).

```
87        · -- **Goal 1 (W ⊆ D)**:
88          -- Let `element` be in W.
89          -- Hypothesis: there exists `x` in C.space, x ≠ 0 such that element = hammingDist x 0
90          -- We use recursive intro, as we have multiple hypothesis, to name them or directly perform
91          -- chosen actions on them (rfl)
92          rintro ⟨x, hx_space, hx_neq, rfl⟩
93          -- Since C.space is a subspace, 0 is in C.space
94          have h_zero : (0 : Fin n → F) ∈ C.space := Submodule.zero_mem C.space
95          -- Hence, `element` is in D.
96          exact ⟨x, hx_space, 0, h_zero, hx_neq, rfl⟩
```

- For proving that the distance set is contained in the weight set, we use that d(x, y) = w(-x + y) (`use -x + y` and `hammingDist_eq_hammingNorm`). Both relations between Hamming weight and distance were already in mathlib.

```
97        · -- **Goal 2 (D ⊆ W)**:
98          -- Analogously, let `element` be in D.
99          rintro ⟨x, hx_space, y, hy_space, hxy_neq, rfl⟩
100         -- We use the codeword `-x + y`
101         use -x + y
102         -- Now we need to prove that:
103         -- 1) -x + y is in C.space
104         -- 2) -x + y ≠ 0
105         -- 3) hammingDist x y = hammingDist (-x + y) 0
106
107         -- 1) C is a subspace, so it is closed under negation and addition
108         have hc_space : -x + y ∈ C.space :=
109           Submodule.add_mem C.space (Submodule.neg_mem C.space hx_space) hy_space
110         -- 2) Since x ≠ y, -x + y cannot be 0
111         have hc_neq : -x + y ≠ 0 := by
112           intro h
113           apply hxy_neq
114           -- `neg_add_eq_zero` states that `-a + b = 0 ↔ a = b`
115           exact neg_add_eq_zero.mp h
116         -- Now we have the first two properties and the goal moves to proving the last one
117         refine ⟨hc_space, hc_neq, ?_⟩
118         -- 3) `hammingDist_eq_hammingNorm` states that `hammingDist x y = hammingNorm (-x + y)`
119         rw [hammingDist_eq_hammingNorm x y]
120         -- The goal is now: `hammingNorm (-x + y) = hammingDist (-x + y) 0`
121         -- `hammingDist_zero_right` states that `hammingDist x 0 = hammingNorm x`
122         exact hammingDist_zero_right (-x + y)
```

- `HammingSphere` : This is simply a sphere in the metric space F_q^n with the Hamming distance. The definition is what you would expect.

```
127     -- A **Hamming sphere** `S_{radius}(center)` is the set of all vectors `v ∈ F_q^n` such that
128     -- `hammingDist(center, v) ≤ radius`
129     def HammingSphere {n : ℕ} (center : Fin n → F) (radius : ℕ) : Set (Fin n → F) :=
130       {v | hammingDist center v ≤ radius}
```

- Theorem `disjoint_spheres` : If `d` is the minimum distance of a code `C` and `t = floor((d-1)/2)` , then the Hamming spheres of radius `t` centered at different codewords are disjoint. More generally, if `d` is a lower bound for the minimum distance.

As stated in the overview, the lemma is used for the theorem, and the theorem is important because it simplifies the task of decoding a received message to eliminate errors introduced during transmission.

```
132     -- **Theorem:** If `d` is the minimum distance of a code `C` and `t = floor((d-1)/2)`,
133     -- then the Hamming spheres of radius `t` centered at different codewords are disjoint.
134     -- (More generally, if `d` is a lower bound for the minimum distance)
135     -- This theorem means that the spheres are pairwise disjoint provided their radius is chosen small
136     -- enough.
137     -- (This is theorem 1.11.2 in the book)
138     omit [Fintype F] in
139     theorem disjoint_spheres {n k d t : ℕ} (C : LinearCodeWithDist F n k d) --The code and the bound `d`
140         -- The condition on `t`
141         (ht : 2 * t < d) -- [⇔ 2t < d ⇔ 2t ≤ d-1 ⇔ t ≤ (d-1)/2 ⇔ t ≤ floor((d-1)/2)]
142         -- The codewords `c₁` and `c₂`
143         (c₁ c₂ : Fin n → F) (hc₁ : c₁ ∈ C.code.space) (hc₂ : c₂ ∈ C.code.space)
144         -- A vector `z` in both spheres
145         (z : Fin n → F) (hz₁ : z ∈ HammingSphere F c₁ t) (hz₂ : z ∈ HammingSphere F c₂ t) :
146         -- Thesis: the codewords are the same, i.e., the intersection of spheres with different centers
147         -- is empty, so they are disjoint
148         c₁ = c₂ := by
```

- Theorem `disjoint_spheres` : We prove this by contradiction (`by_contra`), assuming that the are two distinct codewords such that there exists an element in the intersection of their spheres:

```
149         -- Proof by contradiction: assume h_neq: c₁ ≠ c₂
150         by_contra h_neq
```

i. Apply triangle inequality (`hammingDist_triangle` , already in mathlib) to bound the distance between the two codewords using the intersection element.

ii. Use the definition of the sphere to bound that by `d` .

```
151       -- By the triangle inequality, d(c₁, c₂) ≤ d(c₁, z) + d(z, c₂),
152       -- which is ≤ t + t = 2t since `z` is in both spheres:
153       have h_dist_le_2t : hammingDist c₁ c₂ ≤ 2 * t := by
154         calc
155           hammingDist c₁ c₂ ≤ hammingDist c₁ z + hammingDist z c₂ := hammingDist_triangle c₁ z c₂
156           -- Now we flip the second term using symmetry to match the sphere definition
157           _ = hammingDist c₁ z + hammingDist c₂ z := by rw [hammingDist_comm z c₂]
158           -- `add_le_add` states that if (h₁ : a ≤ b) (h₂ : c ≤ d), then a + c ≤ b + d
159           _ ≤ t + t := add_le_add hz₁ hz₂
160           _ = 2 * t := by omega
```

iii. But two different codewords can`t be closer than the minimum distance of the code because of the lemma!

```
161       -- The minimum distance is ≤ hammingDist c₁ c₂, applying the lemma we proved before
162       -- due to c₁ and c₂ being assumed different
163       have h_minDist_le_dist : minDist F C.code ≤ hammingDist c₁ c₂ := by
164         rw [minDist_eq_min_pairwise_dist F C.code]
165         -- Now the goal is:
166         -- sInf {d_1 | ∃ x ∈ C.code.space, ∃ y ∈ C.code.space, x ≠ y ∧ d_1 = hammingDist x y}
167         --  ≤ hammingDist c₁ c₂
168         -- The infimum of a set of naturals is ≤ any element contained in the set, by `csInf_le`
169         -- `OrderBot.bddBelow` states that when there is a global minimum (such as subsets of naturals),
170         -- every set is bounded below.
171         apply csInf_le (OrderBot.bddBelow _)
172         -- Now we prove that `hammingDist c₁ c₂` is in the set
173         -- It verifies the property:`Set.mem_setOf_eq` states that `(x ∈ {y | p y}) = p x`
174         simp only [Set.mem_setOf_eq]
175         -- And it's the distance between some x ≠ y in C, which are c₁ and c₂
176         exact ⟨c₁, hc₁, c₂, hc₂, h_neq, rfl⟩
177       -- Since `d` is a lower bound for `minDist`:
178       -- `d ≤ minDist ≤ d(c₁, c₂) ≤ 2t`
179       have h_d_le_2t : d ≤ 2 * t := by
180         calc
181           d ≤ minDist F C.code := C.dist_lower_bound
182           _ ≤ hammingDist c₁ c₂ := h_minDist_le_dist -- by the previous step
183           _ ≤ 2 * t := h_dist_le_2t -- by the step before that
184       -- This yields d ≤ 2t, which strictly contradicts ht : 2t < d
185       omega
```

```
187     -- **On the importance of this theorem:** If a codeword `c ∈ C` is sent and a vector `y ∈ F_q^n` is
188     -- received where `t` or fewer errors have occurred (that is, `y = c + e`, `e ≠ 0` and `w(e) ≤ t`),
189     -- then `c` is the unique codeword closest to `y`.
190     -- In particular, nearest neighbor decoding uniquely and correctly decodes any received vector
191     -- in which at most `t` errors have occurred in transmission (Corollary 1.11.3 in the book).
```

Challenges:

The most challenging aspect for me was, as I already described, deciding how to define concepts, since I wanted the definitions to simplify the proofs but I was also wondering about consistency with mathlib and scalability. Finding the right tactics to execute the proofs was also challenging, and searching mathlib for the definitions and theorems.

Note:

I would like to add these definitions and theorems to mathlib, but I still have to check the guidelines to adapt it. If anyone has recommendations or insights, let me know!

The End