Random thesis notes

ISD -> Family of algorithms for solving MLD/Min. distance ISD algorithms can solve the problem given any code C.

ISD algorithms are divided in three steps:

- PGE
- Solving the small instance
- Producing solutions

We talk about iterations as the ISD algorithm can be run multiple times varying the parameter w to obtain a certain result.

PGE

Small instance

At this point we can represent every codeword as the concatenation of two smaller codewords, that respect 2 particular equations. We can play with these equations to obtain a codeword easily. By forcing c' to have a low weight, we can find low codewords easier. The meat and potatoes of HOW TO DO THIS, vary algorithm by algorithm. We will describe the differences that occur between the reknown Stern and Lee & Brickell algorithms. This part produces the codewords in the permutation of the codewords of weight w, such that a specific constraint is satisfied. We will denote by a function f, given a certain codeword whether it respects the constraints or not. If f return 1 the given codeword is added to the output. Crucial quantities:

- success probability
- average codeword found per iteration

Focussing on the success probability, we have to assess the chance that a given permutation pi is succesful for ISD. For random codes this is function of:

- weight w
- constraints imposed by ISD variant

Producing Solutions

Once a part of the codeword is found,the other part is easily computable. We produce codewords in a certain form and check whether they have the desired weight. These codewords are just a permutation of a codeword in Cw.

Data

Data:

- Solve subroutine (varies between algorithms)
- l in N, 1 <= l <= r

Input:

- H code parity check matrix
- w searched weight

Output:

- Y subset of the codewords of weight w

Improvements over Stern

Idea 1

Exploit sparsity of H. We can construct the small instances in a convenient way/ Parameter l impacts:

- success probability decreases as l increases
- list sizes increase with l
- probability of collision decay exponentially with l

This first algorithm is based solely on this approach on sparsity.

Algorithm We partition the matrix `U * pi(H)` Small instance is `l x (k+l)` (again) Now top right submatrix is required to be null.

The partition can be obtained by:

1. select random t (parameter) rows and denote their support with J1,...,Jt
2. unite all the supports to form J (basically list of ones in the t rows). We move all these ones in the leftmost position by selecting a certain pemrutation sigma. At this point the first t rows of H are in the form (A,0). Where A is `t x z`. 0 is `t x (n-z)` null matrix. Basically z = |J|.
3. select another permutation phi in Sn that fixes the first z coordinates and computes H' (in the permutation the first z rows are not moved)
4. perform PGE on the last n-(l-t) rows of H', producing the identity I of size n-(k+l) in the rightmost bottom corner

This corresponds to applying permutation pi=phi°sigma and change of basis U (see article for specifics). We can partition x'=pi(x) in 3 parts (x1,x2,x3), with lengths z, k+l-z, n-(k+l). This produce a linear system.

We perform a 2-steps meet-in-the-middle approach:

- find candidates for x1 splitting A in z/2 columns (first equation)
- enumerate candidates for x2 and apply meet-in-the-middle approach (second equation)
- test the pairs using C matrix (third equation)

If we assume that the solution probability is not affected by the choice of sigma then we can estimate the cost of the algorithm (see article for specifics).