PROGRESS REPORT ON GREEDY ALGORITHMS IN CODING THEORY

Dejan Spasov

Faculty of Computer Sciences and Engineering

Skopje, Macedonia

ABSTRACT

Greedy algorithms in Coding Theory are simple to define and easy to implement, but require exponential running time. Codes obtained with greedy constructions have very good parameters, thus improving the running time of these algorithms may lead to discovery of new codes with best known parameters. We give an overview of greedy algorithms and discuss further improvements.

I. INTRODUCTION

Let F_q^n be n-dimensional vector space over finite field F_q. Let $d(x,y)$ denote the *Hamming distance* between two vectors $x,y \in F_q^n$, and let (n,M,d) be an M-size *code* C over the n-dimensional space F_q^n with minimum distance d. (The *minimum distance* d of a code C is defined as $d = \min(d(x,y))$, $\forall x,y \in C$.)

The main focus in this paper will be on *linear codes,* i.e. k-dimensional linear subspaces of F_q^n. We write $[n,k,d]$ to denote a linear code of dimension k and minimum distance d. A linear code is completely determined with its $k \times n$ generator matrix G or its $(n-k)\times n$ parity check matrix H, such that $HG^T = 0$. Throughout the paper, we will assume that the generator and parity check matrices are in *standard form*, i.e. $G = [I \quad A]$, and $H = [-A^T \quad I]$.

We use $wt(x) \in \mathrm{N}$ to denote the Hamming weight of the vector x, i.e. the number of nonzero positions of x ($wt(x) = dist(x,0)$).

Asymptotically speaking, δ will denote the *relative distance* of the code $\delta = d/n$, and R will be the *code rate* $R = \log_q(M)/n$. A string of n ones, 11…1, will be written as 1^n, and the concatenation of two strings a and b will be represented with $(a \mid b)$.

Fundamental problem in coding theory is how to find *optimal codes*. The code (n,M,d) is optimal if it has maximal number of codewords M for a given n and d. In general, finding an optimal code is considered to be a difficult problem. Trivial way to do this is by super-exponential search over all possible orderings of the field F_q^n. For small fields ($q \le 9$, $n \le 256$), there exist tables of best known (some of them optimal) codes [1], but for larger spaces optimal-code parameters can be estimated with the Gilbert-Varshamov bound and its asymptotical variant.

In estimating the complexity of an algorithm, we adopt Random Access Machine (RAM) as a computational model. The time complexity is measured as the number of basic (sequential) steps needed for instance of the algorithm to end. It is considered that RAM has unlimited memory with instant access. Thus the space complexity is simply the number of registers used by an instance of the algorithm.

II. GREEDY ALGORITHMS

It is well-known that simple greedy search produces a code with parameters that follow the Gilbert-Varshamov bound. Here we give a brief overview of these algorithms.

A. Gilbert Construction

In general, Gilbert's Construction produces a nonlinear (n,M,d) code. Given the length n and the minimum distance d, the algorithm will search the entire F_q^n and greedily adds to C the first vector x, such that $d(x,c) \ge d, \forall c \in C$. The time complexity of this algorithm is $O(nq^{(1+R)n})$, and space complexity is determined by the size of C, i.e. nq^{Rn}.

B. Varshamov Construction

The Varshamov's algorithm [X] produces the parity check matrix H of a linear code. The algorithm greedily adds to H the first vector x that is not linear combination of $d-2$ or less columns of H. The space complexity of this construction limited to the space need to store the parity check matrix , which is quadratic with respect to code length n, i.e. $O((1-R)n^2)$. The time complexity, compared to Gilbert construction, is also improved $O(n^2 q^{n(1-R+H(\delta))})$.

C. *Jenkins' Construction*

The idea in this construction is to build the A matrix of a systematic code $G=[I \quad A]$. Each $x \in F_q^{n-k}$ is inserted as a row in A. Then for each i-linear combinations of rows of A, where $i \le (d-2)$, we check if it has Hamming weight at least $d-i$. The space complexity, determined by the A matrix, is $O((1-R)Rn^2)$ and the time complexity is

$$O\left(n^2 q^{n(1-R+RH(\delta/R))}\right).$$

First we should note that there is slight improvement compared to Varshamov construction, due to the smaller dimensions of the A matrix. Second, the algorithm heavily depends on efficient algorithm for computing the Hamming weight of a vector [2].

D. *Lexicograpfic construction*

Again, in this construction we build the A matrix of a systematic code $G=[I \quad A]$. For each $x \in F_q^{n-k}$ we reserve $\log(n-k)$ space. Thus, in the memory, we create a data structure of size $\log((1-R)n)q^{(1-R)n}$. In the x-th $\log(n-k)$-sized cell of this structure we store the number of rows of A that make a linear combination equal to x. Lexicographic construction is simply a linear search over this data structure, thus the time complexity is $O(\log((1-R)n)q^{(1-R)n})$ [3].

III. Algorithms for Computing Hamming Weight

Since the Jenkins' construction depends on fast algorithms for computing the Hamming weight, in this section we list some of them. We start with one of the most widely used algorithms:

```
Algorithm #1
 x=(x&0x55555555)+((x&0xaaaaaaaa)>> 1);
 x=(x&0x33333333)+((x&0xcccccccc)>> 2);
 x=(x&0xf0f0f0f0)+((x&0xf0f0f0f0)>> 4);
 x=(x&0xff00ff00)+((x&0xff00ff00)>> 8);
 x=(x&0xffff0000)+((x&0xffff0000)>>16);
 return x;
End of Algorithm #1
```

In this setting, it is considered that Algorithm #1 uses 20 $O(n)$ basic operations. The number of basic operations can be reduced to 15 (see Algorithm #2) [4].

```
Algorithm #2
 x-=(x >> 1) & 0x55555555;
 x =(x & 0x33333333) + ((x >> 2) & 0x33333333);
 x =(x + (x>>4))& 0xf0f0f0f0;
 x+= x >>  8;
 x+= x >> 16;
 return x&0x7f;
End of Algorithm #2
```

However, if we precompute and store the weight of each 16-bit or 32-bit word, then the following algorithm will retrieve the hamming weight from the memory in $O(\log(n))$ time

```
Algorithm #3
 char hw_table[65536]; // H. weight storage
 int p=0,*p1=(int *)&p, *p2= p1+1;
 #define wt(x) (p=x, hw_table[*p1]+hw_table[*p2])
End of Algorithm #3
```

IV. Conclusion and Future Work

The algorithm that we use to search for good codes is a combination of the lexicographic construction and the Jennkins' algorithm [X]. With this algorithm we were able to find the following best-known codes: $[68, 50, 9]_4$, $[67, 49, 9]_4$ and $[92, 75, 8]_4$ [1]. The running time of this algorithm is determined by the Jenkins' algorithm, with constant improvement due to the lexicographic construction. For certain (smaller) minimum distances this constant improvement can be very useful to achieve good results. Thus improving the algorithm for computing the Hamming weight is particularly useful in our search for new best known codes. Bellow we compare the performance of the Jenkins' algorithm in case when we use algorithm #1 and algorithm #3. It is obvious that the speed-up is asymptotical. This is expected result even between algorithm #1 and algorithm #2 since Hamming weight function is called exponential number of times. Our future work is to employ algorithm #3 in order to find new best codes.

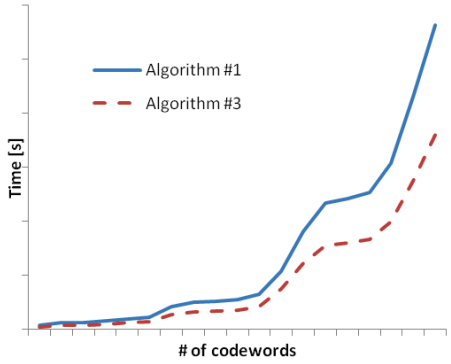

Fig. 1. Running time of the Jenkins' algorithm in case it uses algorithm #1 and algorithm #3

References

[1] Grassl, M. "Bounds on the minimum distance of linear codes and quantum codes." Online available at: http://www.codetables.de.

[2] Jenkins B. "Tables of lexicodes." Online available at: http://burtleburtle.net/bob/math/lexicode.html.

[3] Spasov, D. "Some properties of good codes". PhD Thesis, Ss. Cyril and Methodius University, Skopje, 2010.

[4] Wikipedia. "Hamming weight." Online available at: http://en.wikipedia.org/wiki/Hamming_weight.