Supplementary Materials: Introduction of Algorithmic Complexity and Coding Theory

Hoover H. F. Yin, Ka Hei Ng, Shi Kin Ma, Harry W. H. Wong and Hugo Wai Leung Mak

This supplementary materials provide a brief introduction of algorithmic complexity and coding theory for readers who are not familiar with these topics.

1. Introduction of Algorithmic Complexity

It takes time for a computer to conduct every operation, e.g., addition, multiplication, etc. An *algorithm* is a finite sequence of operations (which can be directly computed by the computer) to solve a problem or perform a computation. The non-constant parameters of the problem are the inputs of the algorithm, where the input size is measured in bits (the number of binary symbols). The number of operations needed to solve a problem by an algorithm can be expressed as a function of the input size. That is, this function can be regarded as a measure of the time required in solving the problem, with the use of such algorithm. When the input size is significantly large, the running time, which is dominated by the dominant term of the function, to solve the same problem by two different algorithms can be significantly differed.

In this appendix, we roughly describe the expression of time complexity in big O notation, and the complexity classes P, NP, NP-complete and NP-hard. For a more rigorous discussion, we refer readers to computer science textbooks such as [31,32].

1.1. Complexity Class P

To describe the efficiency of an algorithm, *big O notation* is used in the context of algorithmic complexity. Let n be the input size, $f(n)$ be the number of operations needed to solve the problem by the algorithm, and $g(n)$ be the comparison function for estimating $f(n)$. We write

$$f(n) = \mathcal{O}(g(n))$$

if there exist positive integers c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$. Then, we say that the algorithm solves the problem in $\mathcal{O}(g(n))$ time. For example, let $f(n) = 5n^3 - 4n^2 + 3n - 2$. As n is a positive integer, we have $f(n) \leq 5n^3 + 4n^3 + 3n^3 + 2n^3 = 14n^3$. Therefore, we know that $f(n) = \mathcal{O}(n^3)$, where the dominant term n^3 is extracted by the big O notation.

If the algorithm runs in $\mathcal{O}(g(n))$ time, where $g(n)$ is a finite-degree polynomial expression in n, then we say that the algorithm is a *polynomial-time* algorithm. Roughly speaking, the complexity class P consists of all *decision problems*, i.e., yes-no questions, which can be solved by some polynomial-time algorithms. A more rigorous definition involves the definition of deterministic Turing machine, which is out of the scope of our discussion in this paper. As a rule of thumb, the problems in P are considered as "efficiently solvable problems".

1.2. Complexity Class NP

The computer that we considered in the above context can only run operations one by one. That is, when we want to compute multiple branches, we can only compute them in a step-wise manner. A figurative example is that we are solving a maze - at a certain point, we need to determine whether turn left or turn right. In principle, we will need to try both ways to ensure that we can find an exit. That is, we have to come back to this position and try the other way later. Yet, if we have a multicore processor, we can try both ways at the same time in different cores. Suppose the processor has infinitely many cores, and we can solve a problem in polynomial time (in parallel), then we say that the problem can be solved by an algorithm in *non-deterministic polynomial time*. Roughly speaking, the complexity

class NP is the collection of all decision problems that can be solved in non-deterministic polynomial time. Similarly, a more rigorous definition involves the non-deterministic Turing machine, but in this context, we omit such an in-depth discussion.

We can regard NP as a class of decision problems that can be verified in polynomial time. For example, suppose we obtain a path in a maze, and we want to verify whether this path could lead to the desired exit. What we need to do is to follow this path and see if we can really reach the exit. This can be done in polynomial time, because this path was generated by one of the cores in polynomial time when solving the maze. Note that P is a subclass of NP, because we can verify the answer in polynomial time by solving the problem as well.

1.3. Complexity Class NP-Complete

In mathematics, we usually transform a problem to another one that we know how to solve. This is known as *reduction*. When we reduce a problem A to a problem B, it means that problem B covers all the instances of problem A. If the reduction can be done in polynomial time, we say that A is reduced to B in polynomial time. In other words, if we can efficiently solve problem B, then we can also efficiently solve problem A with a polynomial-time reduction. However, the converse is not always true. Although it is called "reduction", we are actually transforming a problem to a "harder" problem.

The hardest type of problems in NP is called *NP-complete* problems. That is, any problem in NP can be reduced to an NP-complete problem in polynomial time. If we can solve any of the NP-complete problems in polynomial time, then we can solve all NP problems in polynomial time, which implies that P equals NP. However, the existence of such a polynomial-time algorithm is still unknown. Proving or disproving the existence of such algorithm can actually solve the "P versus NP problem", which is one of the Millennium Prize Problems in Mathematics [33]. The answer of this problem has crucial consequences and implications in different branches of mathematics and computer science [34,35]. Throughout recent decades, it is widely believed that no such algorithm exists [36–38], but it is yet to be proven.

1.4. Complexity Class NP-Hard

A problem that is reduced from an NP-complete problem in polynomial time is called an *NP-hard* problem. In other words, an NP-hard problem is no "easier" than any NP-complete problems. Note that an NP-hard problem may not be in NP, i.e., we may not be able to verify the answer in polynomial time. If an NP-hard problem is in NP, then the problem is NP-complete, and this is actually a standard technique to prove the NP-completeness of a particular problem. A convention is that the class "NP-hard" is not restricted to decision problems. If the decision problem (e.g., does a solution exist?) is NP-complete, then its associated function problem, i.e., finding an instance of the solution, is NP-hard. This is because if we obtain a solution, we can also answer the decision problem, thus the function problem is no easier than NP-complete.

2. Crash Course on Coding Theory

We focus on describing the background of coding theory that is related to this paper. As a convention, when we say coding theory, we refer to *channel coding theory*, which is a study of error correction. We refer the readers to textbooks such as [39,40] for a more comprehensive discussion of coding theory.

2.1. Error-Correcting Codes

One of the main goals in coding theory is to send a message (represented by a sequence of symbols) from a source to a destination via a "noisy" medium that could introduce "errors" to the message. A common form of "errors" is to modify some symbols in the message. To ensure that the destination can accurately and effectively recover the message, we need to add redundant information to the message to form a *codeword* of an *error-*

correcting code. Due to some practical considerations, such as the way to distinguish different codewords, most error-correcting codes, e.g., [41,52], have assumed that every codeword is of the same length. Every possible sequence that has the same length as a codeword is called a *word*.

The overall picture is that the source transforms a message m into a codeword c, then transmits c to the destination. The destination receives a word $\hat{c}$, which may be different from c due to errors. Then, the destination tries to guess the correct codeword c, then transforms it back to the message. It is easy to visualize that some constraints must be imposed on the number of errors, otherwise there is no clue to guess the correct codeword in a systematic and scientific manner.

In convention, we use *Hamming distance* as the metric to measure the number of errors.

Definition 1 (Hamming Distance)**.** *The* Hamming distance *between two sequences of the same length is defined as the number of symbols at which the corresponding values are different.*

Let n be the length of any word, and $q \geq 2$ be the size of the alphabet used by the words. That is, there are totally q^n possible words. For each possible codeword c, let $\text{Ball}_r(c)$ be the set that contains every word w such that the Hamming distance between w and c is at most r. In other words, $\text{Ball}_r(c)$ is the closed *Hamming ball* of radius r centered at c. By direct counting, the volume of $\text{Ball}_r(c)$ is

$$\text{Vol}(\text{Ball}_r(c)) := \sum_{i=0}^{r} \binom{n}{i} (q-1)^i.$$

Definition 2 (Distance of Codes)**.** *The* distance *of an error-correcting code, denoted by d, is defined as the minimum Hamming distance between any pair of distinct codewords.*

As long as the Hamming balls $\text{Ball}_r(c)$ of all possible codewords c do not overlap, when we receive a word w, we say that the correct codeword is c if $w \in \text{Ball}_r(c)$. To maximize the number of correctable words, we maximize the radius of the balls as long as they do not overlap with each other. The distance of the code, d, is either $2r+1$ or $2r+2$. In other words, such code can guarantee to detect at most $d-1$ errors, and correct at most $\lfloor \frac{d-1}{2} \rfloor$ errors.

The aforementioned decoding approach is called the *minimum distance decoding*, or the *nearest neighbor decoding*, which finds the closest codeword by modifying the fewest number of symbols in the received word. A *binary symmetric channel (BSC)* is a channel model such that every symbol (0 or 1) has the same (crossover) probability to be modified into another symbol. For a BSC with a crossover probability of less than 0.5, the minimum distance decoding process is equivalent to the *maximum likelihood decoding* process, which aims at finding a codeword that has the maximum likelihood.

Definition 3 (Covering Radius)**.** *The* covering radius *R is the smallest r such that all words are covered by the union of all codeword-centered Hamming balls of radius r.*

In other words, any words must have a codeword within R Hamming distance. The covering radius is applied in coding for write-once memories [42], football pool betting [43], etc.

2.2. Hamming Bound & Singleton Bound

Given q, n and d, the maximal number of codewords among all possible codes is denoted by $A_q(n,d)$, and the code having $A_q(n,d)$ codewords is called an *optimal code*. The method to find the exact value of $A_q(n,d)$ for arbitrary q, n and d remains to be an open problem, and is known as the *main coding theory problem*. Nevertheless, many bounds of $A_q(n,d)$ have already been established. Some of the bounds are rexpressed in the form of an inequality, e.g., the Plotkin bound [44] and the Gilbert-Varshamov bound [45,46],

while some of them are expressed in the form of an optimization problem, e.g., the linear programming bound [47] and the semidefinite programming bound [48].

One of the earliest bounds is the *Hamming bound* [52]. The idea is straightforward: The total volume of all non-overlapping Hamming balls cannot excess the number of all possible words. We can ensure the non-overlapping constraint when the radius of every Hamming ball centered at a codeword is no larger than $\lfloor \frac{d-1}{2} \rfloor$. Mathematically speaking,

$$A_q(n,d) \leq \frac{q^n}{\sum_{i=0}^{\lfloor \frac{d-1}{2} \rfloor} \binom{n}{i}(q-1)^i}. \qquad \text{(Hamming Bound)}$$

A code such that the equality case of the Hamming bound holds is called a *perfect code*. That is, in a perfect code, every word is covered by one and only one codeword-centered Hamming ball of radius $\lfloor \frac{d-1}{2} \rfloor$, thus every word can be corrected as a unique codeword. However, Tietäväinen [49] has proven the non-existence of perfect codes over a prime-power alphabet except trivial perfect codes (the code distance is either 1 or n), Hamming codes (widely used in error correction code (ECC) memory), and Golay codes (used by the NASA's Voyager [50]).

Another bound that we used in this paper is called the Singleton bound [26]. The idea of this bound is that, if we delete the first $(d-1)$ symbols of every codeword, i.e., the length of every codeword is now $(n-d+1)$, then each pair of altered codewords is still separated by a Hamming distance of at least 1. The number of altered codewords, which is the same as the number of original codewords, is at most q^{n-d+1}. Mathematically speaking,

$$A_q(n,d) \leq q^{n-d+1}. \qquad \text{(Singleton Bound)}$$

A class of codes known as the *maximum distance separable (MDS) codes* achieves the equality of the Singleton bound. Reed-Solomon code [41] is a type of MDS code that has been applied in many modern technologies, including CDs, QR codes, etc.

2.3. Linear Codes

A *linear code* is a vector space. Linear codes are of interests in the community of coding theory, as we can apply a wide range of tools in linear algebra. However, this also means that we have restricted the alphabet size to a prime power, as the vector space is defined over a finite field. Further, the number of codewords, i.e., the size of the vector space, must be a power of q. Given q, n and d, the maximal number of codewords among all possible linear codes is denoted by $B_q(n,d)$, and the linear code having $B_q(n,d)$ codewords is called an *optimal linear code*. Note that $B_q(n,d)$ must satisfy

$$B_q(n,d) \leq q^{\lfloor \log_q A_q(n,d) \rfloor}.$$

Definition 4 (Hamming Weight)**.** *The* Hamming weight *of a codeword c, denoted by* $\mathrm{wt}(c)$*, is the Hamming distance between c and the zero vector, i.e., the number of non-zero symbols in c. The* Hamming weight *of a code is the minimal Hamming weight among all the non-zero codewords of this code.*

Let u and v be two codewords such that the distance in between is d. As a vector space, the linear combination of codewords is also a codeword. Therefore, $u-v$ is also a codeword. Note that $\mathrm{wt}(u-v)=d$. In other words, the distance of a linear code equals to the Hamming weight of the code. Despite the simplified criteria, finding the Hamming weight of a code is an NP-hard problem [16].

In the following, we express the message m and the codeword c as row vectors. To encode m, we calculate $c=mG$, where G is called the *generator matrix* of the linear code. The generator matrix is associated with a *parity check matrix* H such that $GH^\intercal$ gives a zero matrix. In standard form, G is written as a block matrix $(I \mid P)$ for some matrix P and

identity matrix I. Thus, we can write $H = (-P^\intercal \mid I)$. Note that the identity matrices in G and H may be of different dimensions.

Although encoding is easy, decoding is generally an NP-hard problem [7,8], unless we can exploit some special structures of the code, e.g., as described in [51]. For linear codes, we can extract some information regarding the error pattern. In particular, let w be the received word. We can express $w := c + e$ for some error e from a codeword c. Recall that $GH^\intercal$ gives a zero matrix, therefore we have

$$wH^\intercal = (c+e)H^\intercal = mGH^\intercal + eH^\intercal = eH^\intercal,$$

which is known as the *syndrome* of w, and such syndrome identifies the error pattern. In other words, if we pre-compute the syndromes for all possible correctable error patterns, i.e., the syndromes of all e with $\mathrm{wt}(e) \leq \lfloor \frac{d-1}{2} \rfloor$, then we have a lookup table of size $\sum_{i=0}^{\lfloor \frac{d-1}{2} \rfloor} \binom{n}{i}(q-1)^i$ for mapping a syndrome to e. We can correct a word w by $w - e$, where e is obtained by looking up the syndrome of w from the table, i.e., $wH^\intercal$. This approach is known as *syndrome decoding*.

Further, let C be the abelian group of all codewords under vector addition. As C is a normal subgroup of $\mathbb{F}_q^n$ under vector addition, we have the quotient group $\mathbb{F}_q^n / C = \{C + e \colon e \in \mathbb{F}_q^n\}$. Every entry in the quotient group is a coset.

Definition 5 (Coset Leader)**.** *The word that has the minimal Hamming weight in each coset is the* coset leader *of this coset.*

That is, the coset leader is the minimum-weight error pattern that leads or maps a codeword to a word in this coset. We can view the decoding procedure of a word in the coset as subtracting the given word by the coset leader of the coset.

At last, we describe the *shortening* technique, which is a propagation rules of modifying linear codes. Shortening means that for any linear code of codeword length ℓ, dimension k (i.e., the number of codewords is q^k) and code distance d, there exists a linear code of codeword length $\ell - x$, dimension $k - x$ and code distance d for any $1 \leq x \leq k-1$. This can be achieved by removing some columns of the parity check matrix. The proof of shortening can be found in standard textbooks on coding theory such as [40].

2.4. Repetition Codes, Hamming Codes, and Extended Hamming Codes

For simplicity, we only consider binary field in this sub-section.

Repetition code is one of the simplest codes that repeats a single bit to be encoded into a length n codeword. That is, there are totally two codewords in this code. To decode a word, we look for the symbol (0 or 1) that dominates the bit string. If there are more 0s than 1s, then we decode to the bit 0, and vice versa. The distance of the code is n, therefore every repetition code achieves the equality of the Singleton bound, thus being an optimal code.

Hamming code [52] is a perfect code, i.e., achieves the equality of the Hamming bound, thus it is an optimal code. Let $r \geq 2$. A (binary) Hamming code is a linear code with codeword length $2^r - 1$ and code distance 3, with a parity check matrix whose columns consist of all the non-zero vectors of $\mathbb{F}_2^r$. The dimension of the code (i.e., vector space) is $2^r - r - 1$, hence the number of codewords is $2^{2^r - r - 1}$. As the distance is 3, the Hamming code is exactly equivalent to a single error correcting code. An example of the parity check matrix for the case of $r = 3$ is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

It is remarked that the order of the columns are not fixed. However, if the columns are arranged in the order of increasing binary numbers, i.e., the j-th column is the binary representation of the number j, then the syndrome can directly indicate the location of the

error. More precisely, if the syndrome is the binary representation of the number j, then the error is located at the j-th bit.

Extended (binary) Hamming code is a Hamming code by appending a parity check bit. If H is the parity check matrix of a Hamming code, then the corresponding extended Hamming code has a parity check matrix

$$\left(\begin{array}{ccc|c} & & & 0 \\ & H & & \vdots \\ & & & 0 \\ \hline 1 & \cdots & 1 & 1 \end{array}\right).$$

This code is a linear code of codeword length 2^r and code distance 4, thus it can correct exactly one error. As the extra bit is only a parity bit, the number of codewords remains the same as the Hamming code. As a result, the dimension of the code is $2^r - r - 1$, and the number of codewords is 2^{2^r-r-1}.

Abbreviations

The following abbreviations are used in this supplementary materials:

P	polynomial time
NP	non-deterministic polynomial time
BSC	binary symmetric channel
ECC	error correction code / error-correcting code
MDS	maximum distance separable