Simulating a Binary Symmetric Channel

Information Theory Lab 4

Objective

Understand the model of a Binary Symmetric Channel, and simulate a BSC by randomly introducing bit errors in a file.

Theoretical notions

A Binary Symmetric Channel has the following representation:

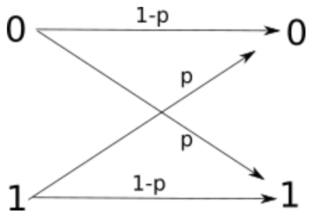

Figure 1: Binary symmetric channel (BSC)

With probability p, a bit will undergo an error, and with probability $1 - p$ it remains the same. Thus, p is known as *the probability of error*.

Practical issues

A data file is a sequence of bits (0/1).

Transmitting a data file over a BSC means that every bit in the original file has a chance p of undergoing an error.

Exercises

1. Write a C program to simulate a BSC for a given file. The program shall be called as follows:

   `BSC.exe 0.01 input.txt output.txt`

- The arguments are:
  - `0.01`: the error probability p of the channel
  - `input.txt`: the input file
  - `output.txt`: the output file
- The program will follow the following steps:
  - Open the first file for reading, and the second file for writing
  - Read every byte value from the input file
  - For every single bit of the byte read, decide whether to change it or not:
    - generate a random number `x`, and based on `x` do the following:
    - toggle the bit, with probability p
    - leave the bit unchanged, with probability $1 - p$
  - Write the resulting byte to the output file

Implementation hints

- The following C functions may be used for file-based operations. Look up their documentation on the Internet (e.g. *cplusplus.com*, or Google search).
  - `fopen(...)`, to open a file for reading;
  - `fread(...)`, to read byte data from the file;
  - `fclose()`, to close the file when finished.
- The following macros implement the basic bit operations:
  - reading a single bit i from a number x;
  - set bit i from a number x to 1;
  - clear bit i from a number x (i.e. set to value 0);
  - change the value of bit i from a number x (i.e. if 0 make 1, if 1 make 0).

```
#define READ_BIT(x,i)      ((x) & (1U << (i)))
#define SET_BIT(x,i)       ((x) = (x) | (1U << (i)))
#define CLEAR_BIT(x,i)     ((x) = (x) & ~(1U << (i)))
#define TOGGLE_BIT(x,i)    ((x) = (x) ^ (1U << (i)))
```

- For randomly deciding when to make an error, with error probability p:
  - use `srand()` once, at the beginning of the program, to seed the random number generator
  - use `rand()` to generate a random number `x` in the range [0 ... RAND_MAX]
  - `x` has p % chances to be smaller than $p \cdot RAND\_MAX$

- therefore: if $r < p \cdot RAND\_MAX$, then change bit; otherwise, leave bit unchanged

Final questions

1. TBD
2. TBD