Algorithms

Hamming Distance Project [100 pts]

1 Introduction

The objective of the project is to (1) implement the greedy-heuristics and exhaustive approaches to solving the Minimum Hamming Distance Problem, (2) measure their run times, and (3) compare these results with their asymptotic complexities. Your program should assume the input file as follows:

```
7
5
10110
11110
11000
10100
00110
11011
11111
```

The first line contains k the length of binary strings is n the number of. Your output should return an integer, the minimum maximum hamming distance and the corresponding binary string. It should be formatted as follows:

```
2
11110
```

Where the first line is the minimum max hamming distance, and the second line is the corresponding binary string.

Note: there may be multiple strings that satisfy the problem, any one of them will do, so long as they are correct. Additionally the optimal binary string may or may not be an element of the given set.

The two solution approaches were discussed in a handout, worksheet, and youtube video.

1. The first approach is the exhaustive algorithm, you need to generate all possible binary strings of length k and calculate the hamming distance for each of those and each of the strings in the given set, and you return the minimum max hamming distance and the corresponding string.
2. Initialize an array of zeros of length k. Iterate through the given set of binary strings and add the value at all indices at the corresponding index in the array. Once done divide each value by n, if the value is less than 0.5 set it to 0, otherwise, set it to 1. The values in the array are now the values for a binary string of length k. Calculate the hamming distance between this new string and all values in the given set, and return the maximum value and the string.

2 Deliverables (consult the rubric on Canvas for this project)

2.1 Report

1. [15 pts] Explain the details of your two implementations. Specifically, in both cases, discuss how you *efficiently* implemented high-level "english" statements provided in the pseudo-code. *Please include a listing of your code in an appendix.*

2. [15 pts] Determine the worst-case time complexity of your algorithms in terms of n and m. (This will depend on your implementation.)

3. [20 pts] Use a random number generator to devise inputs for your algorithms for **at least** four different values of n and m. The values of n and m may need to be different for the two approaches and should be chosen with the following in mind:

   (a) n and m should be large enough so that you can reliably determine the runtime of your algorithm by using an appropriate timing function call such as clock() to time your program; i.e., the minimum run time should be well above the lowest unit measured by your clock function.

   (b) Also choose n so that you can experimentally verify the theoretical runtime you derived above.

   For each n, determine the run time by taking the average of three runs on the same input. This reduces the likelihood of inaccuracies due to system load. Display your results in a table. Explain your choice of n.

4. [10 pts] Match theory and practice: Argue/demonstrate that your experimental runtimes are consistent with the theoretical complexities you derived.

2.2 Online Verification on Canvas

[40 pts] Verification: you will need to run both of your algorithms on inputs supplied by us. *Please ensure that your exhaustive algorithm is able to run when $k = 25$ (it might take a few minutes).*