E2 Mastermind/CodeBreaker

E2.1 Mastermind or CodeBreaker

CodeBreaker or Mastermind is a game played between two players, **Maker** and **Breaker**. The Maker chooses a secret and the Breaker asks questions about the secret in the form of guesses; Breaker’s goal is to determine the secret in as few guesses as possible. The secret is a row of colored pegs, as is each guess. In the original version a row contained 4 pegs (or holes) and there were 6 colors; so there were 1296 possible secrets.

For each guess, the user is given two numbers, **Black** and **White**: Black is the number of positions where the two rows coincide, and White is the number of the Breaker’s pegs that are “correct but in the incorrect position”.

We will use digits to represent the colors. So for example, suppose the secret is 5215. Then the guess 3265 would receive the response Black=2 and White=0, the guess 5553 would receive the response Black=1 and White=1, and the guess 1526 would receive the response Black=0 and White=3.

E2.2 A Computer Program for Maker

The hard part in producing a computer program to act as Maker is creating a function `compare` that calculates the responses. Other than that, the Maker needs to choose a secret at random, then loop, prompting the user for a guess, and providing feedback until the secret is guessed.

Several papers have been written on how to program the `compare` function. Calculating the Black value is easy: it’s just the number of matches between the two rows. The first step to calculate the White value is to discard all the matches. After that, one can count how many of each color is on each side, adding the minimum of these two numbers to a running total. Another way is presented in the code here. The program compares every peg in the one row with every peg in the other row; if they match they are discarded to prevent re-counting.

`codeBreaker.m, compareCodes.m`
E2.3 A Computer Program for Breaker

It is hard to simulate with a computer program the deductions that a human goes through as a Breaker. In contrast, a computer can do calculations easily.

So one strategy is called the **random feasible strategy**. This means that each time the program picks a guess at random from the remaining feasible (possible) secrets. A natural way to program the random feasible strategy is to generate at the beginning the entire list of all guesses. Then, at each step prune the list by discarding those that can no longer be the secret. That requires code to generate all vectors of a given length with a given range for each position.

```
codeFinder.m
```

A better idea for a computer program is to choose each round the guess that best splits up the remaining feasible set. For example, if we care about worst case performance, then it is natural to take the guess that minimizes the maximum subset remaining. Knuth showed that this strategy guaranteed solving the original (6, 4) game in 5 guesses. For example, in the (6, 4) game there are essentially 5 different first guesses.

<table>
<thead>
<tr>
<th>Guess</th>
<th>Maximum remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXXX</td>
<td>625</td>
</tr>
<tr>
<td>XXXY</td>
<td>317</td>
</tr>
<tr>
<td>XYY</td>
<td>256</td>
</tr>
<tr>
<td>XYYZ</td>
<td>276</td>
</tr>
<tr>
<td>XYZT</td>
<td>312</td>
</tr>
</tbody>
</table>

One way for the computer in general to play is to choose the guess that yields the “most” information. This can be calculated as some norm on the split, or entropy.