Testing - an odd optimization problem
Cap'n Robert Merkel
A-ha Me Hearties????

Why pirates???

Because we're going to go searching for buried treasure!
The search

- A chest of buried treasure somewhere on the island
- No X on the map...
The rules

• One treasure chest
• Known size, shape, and orientation
• No information about location
  – equally likely to be anywhere on the island
• Only way to search – dig a hole.
• Minimize expected # of holes required.
  – The F-measure (because each failed attempt equals a flogging by the captain).
Plan #1

• Cap'n Rrrrt
  1. Choose a spot randomly.
  2. Dig there.
  3. If treasure found, stop,
  4. otherwise, back to step 1
Plan #2

- Captain Aaaaaart
  1. Choose \( n \) possible candidate places to dig.
  2. Choose the candidate \( c \) with the greatest distance from the nearest existing hole (maximin criterion)
  3. Dig at location \( c \)
  4. If treasure found, stop
  5. Otherwise, back to step 1.
Results

• Plan B - ~40% fewer holes than plan A.
• But what about Plan C, D, E...
• Tried many.
• Supplies of rum ran tragically low.
• Some of them were lower-overhead than plan B.
• Results were roughly the same.
Why???

- Were we too busy drinking rum and chasing wenches?
- A more fundamental problem?
Mathematics to the rescue
An Optimal Strategy
An Optimal Strategy
An Optimal Strategy
Random vs. Optimal

• Random F-measure
  – area of treasure is $a$
  – area of island is $A$
  – F-measure for random is $A/a$

• Optimal (and yes it is optimal)
  – $A/a$ test cases
  – On average, hit treasure half way through
  – F-measure is $A/2a$

• Captain Aaaart's strategy not far off optimal!
In case it's not obvious

• Island == input domain of software
• treasure chest = "failure region"
• Result still holds if multiple failure regions, n dimensions etc.
• Also holds if input domain modeled as discrete rather than continuous.
• If we're going to improve testing we need to change assumptions!
What is the ultimate goal anyway?

• Not digging for buried treasure!
• Multiple faults within input domain.
• Lead to multiple failure regions.
• Ultimate goal (Littlewood et al) – improve reliability as much as possible after faults detected in testing are fixed.
• Fiendishly hard to model 😞
Improving failure detection

• Incorporate guess where failures are most likely.
• Add some clues to the treasure map...
Failure-proportional sampling

- Discrete (and large) input domain, $k$ inputs $i_1, i_2, \ldots, i_k$
- Prior probabilities for failure $p_1, p_2, \ldots, p_k$
- Select randomly with replacement.
- Assign selection probability $s_i = \text{failure probability } p_i$
- Sounds like a good idea, right?
Optimal strategy

• Turns out to be no improvement on uniform random selection.
• Optimum strategy = $s_i = \sqrt{p_i}$
• Strategy came from Press(2009). Paper was about looking for terrorists.
Combining locality and probability

- Locality on its own -> 50% improvement
- Probability on its own -> not so useful either
  - Leads to repeatedly hitting high-probability areas.
- Need to combine them.
- Essentially, trying to have a formal mathematical model of debug testing
- But...modelling this is *really* hard.
## The brute force model

<table>
<thead>
<tr>
<th>i1</th>
<th>i2</th>
<th>i3</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>P1</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>P2</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>P3</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>P4</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>P5</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>P6</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>P7</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>P8</td>
</tr>
</tbody>
</table>
The brute force model

- Represents our prior beliefs about failure behaviour
- Can calculate our current beliefs about program reliability.
- In practice, table is intractably huge (2^{input domain}, where input domain is already huge)
- Not obvious what we’d do w/information to deliver reliability improvements.
- Despite size, doesn’t represent everything we’d like to model 😞
Mistakes, failures and faults

• Mistakes (brain fart) -> fault (code fart) -> failure (output fart)

• To improve delivered reliability, fix the faults which cause the most failures.

• Need to incorporate in the model?
  – But model is already intractable!
So...I'm kinda lost