Introduction

The 12-coin problem can be generalized to N-coins. For this project, I looked at how to represent the problem space and write heuristics functions to find the errant coin. Briefly, I take the approach that a jury might take in a legal case. Each weighing results in some evidence and accusations are assigned to each coin: it may be a lighter coin, or it may be a heavier coin, or it definitely is a fair coin. These accusations follow each coin through the weighing trials, and eventually a coin gets more accusations of some kind than any other coin, and it is deemed to be guilty of being lighter or heavier. The heuristics are in deciding which coins to weigh against which other coins. I tried to come up with a best heuristic manually, and tried two random heuristics.

Design

In my design, input is provided as as list of values -1, 0 or 1. Only one item in the list may be non-zero, and its position in the list represents the unfair coin that is lighter (-1) heavier (1). Coins are represented as objects with a history of accusations. Accusations may be a list of integers -1 or 1, created by successive weighing trials, or a singleton 0 indicating a fair coin. Once a coin is deemed fair, it cannot be accused of being lighter or heavier. The coins are kept in a set object called a CoinBag. I can query the CoinBag to decide if a single coin has been accused enough to be found guilty.

The CoinBag object is passed to a heuristic function that performs a single measurement and returns a guilty coin or a negative result indicating that more weighing trials are needed. A limitation in the design is that the heuristic function observes a CoinBag knowing only the accusations given to each coin – it does not know which coins were measured previously. This limitation is intentional as it simplifies the design of the heuristic functions.

Heuristic Functions

For all heuristic functions, the number of coins on each side of the scale is equal. Other configurations result in ambiguous evidence because the scale has only three states and there is no way to compare two independent "less than" measurements.

The hand crafted heuristic function performs by branching into three conditions:

1. If this is the first measurement, all coins are unknown so an initial measurement is made. The number of coins to be placed on each side of the scale is calculated by the Python expression `int(math.ceil(number_of_coins/3.0))`. The measurement between two groups of coins results in evidence, which plays a role in the accusations that are described below.

2. If only one coin remains accused of being heavy or light and the others are all fair, then a simple determination can be made.

3. Otherwise, a similar division of coins is made as in the first case. The difference is three fold:

   (a) The coins to be placed on the scale are all uncertain coins. That is, those that have not been vindicated as being fair coins and so may be unfair.

   (b) The coins not on the scale are the remainder of the above group, augmented with all the fair coins. The size of this group is truncated to the size of the group that will be placed on the scale.

   (c) Before measurement, the coins are shuffled. Several coins that were determined to remain off the scale replace coins on the left side of the scale. This occurs until all slots on the left side are replaced or all remaining coins are used up.

   The result of these selections and movements is that fair coins have a chance to contribute to the disambiguation of fair coins from unfair coins. If a fair coin is weighed on the same side as an unfair coin, then the scale will tip as a result of the single unfair coin.

The evidence after a measurement results in accusations. For example, in the case of N=5 (coins labelled A, B, C, D and E), if coins A and B are weighed to be heavier than coins C and D ("AB > CD"), then the following possible relations exist: A>C, A>D, B>C, B>D. As a result, the accusation that A is heavy (Ah) and B is heavy (Bh) can be made with two votes each. Likewise, C is light (Ch) and D is light (Dl) with two votes each. The result is ambiguous because no coin has more votes than another, so more measurements are necessary. In addition, coin E is determined to be fair because it did not participate in a measurement that contains the unfair coin. On the other hand, if coin C has already been deemed to be fair, accusations of C are not made, and the next weighing trial will consider only three coins instead of four. This continues until a coin receives more votes of being heavier or lighter than the others.

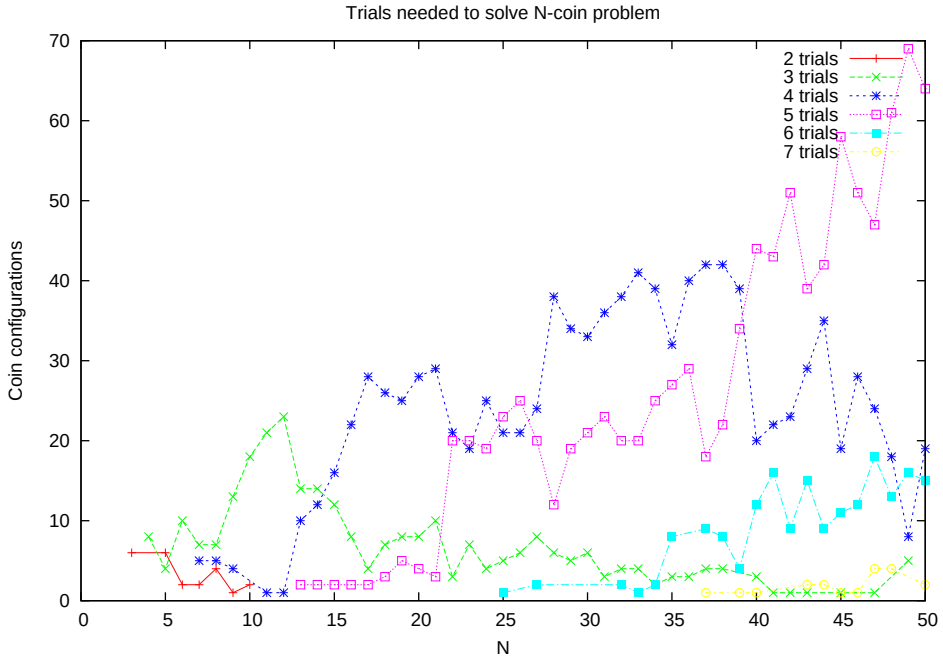
In addition to this, I made two other heuristic functions which choose at random which coins to place on the scale. One selects two coins, the other selects a random size at most half in size as the number of coins. The purpose was to verify that the voting system converges on the correct answer given random evidence.

Testing and Results

To test the heuristic function, I designed input 2N sets for each N-coin problem where N>3. (The case N=1 is degenerate and N=2 is unresolvable because there can be no consensus fair coin.) I measured the number of weighing trials needed and aggregated the number by how many coin configurations required so many trials.

N	result
3	2 trials for 6 configurations
4	2 trials for 1 configuration, 3 trials for 7 configurations
5	2 trials for 5 configurations, 3 trials for 5 configurations
6	2 trials for 2 configurations, 3 trials for 10 configurations
7	2 trials for 4 configurations, 3 trials for 6 configurations, 4 trials for 4 configurations
8	2 trials for 3 configurations, 3 trials for 8 configurations, 4 trials for 5 configurations
9	2 trials for 3 configurations, 3 trials for 12 configurations, 4 trials for 3 configurations
10	2 trials for 4 configurations, 3 trials for 15 configurations, 4 trials for 1 configuration
11	3 trials for 18 configurations, 4 trials for 4 configurations
12	3 trials for 21 configurations, 4 trials for 3 configurations
...	...

This table (extended to N=50) may be seen visually:



It is clear that as the number of coins increases, so does the number of longer trials necessary to find the unfair coin. For both random heuristic functions, the number of trials needed varies and is almost always greater than the hand crafted function.

Discussion

The simple design of this system offers no hindsight for the heuristic function. That is, it cannot know what comparison was made earlier and select coins that will maximize the utility of the measurement. Further, it is possible that a measurement which results in an ambiguous vote still has a stronger logical conclusion based on the history of previous comparisons. For example, if E is deemed to be a fair coin, and the measurement E>A is made, coin A will receive a "A is lighter" vote. But the stronger conclusion would be that "A is lighter than a fair coin, so it must be the lighter unfair coin". It is possible that voting for

"A is lighter" will put A's votes in balance with other coins instead of short circuiting the affair and terminating early. This accounts for the cases where N=12 and 4 trials are needed for three configurations of coins.

Thoughts on earlier attempt

I made an earlier attempt at describing the problem space as a graph. Each node would represent a coin and edges would represent relationships between coins that were established by a single act of weighing. In particular, an edge would be drawn between two coins if they were deemed to be of equal weight (fair coins.) This meant that the graph started without edges and slowly edges were added until one node was isolated. However, this proved to be difficult to conceptualize, because inequality relationships were not described. I stopped short of a multi-graph with many directed edges pointing to heavier nodes. A graph structure could maintain relationships between individual coins, but trials give ambiguous relationships: each "heavier than" relationship is exclusive of the others. Eventually I settled on the voting jury scheme described earlier.

Implementation and data

The code to do this is implemented in Python and provided in the `code.tar.gz` archive. The `coins.py` script drives the operation and the `heuristics.py` script has the heuristic functions: `scale_1` (the hand coded function), `scale_random_pairs`, and `scale_random_groups`. `coins.py` is hard coded to use `scale_1`, but can be modified easily. Other heuristics can be modelled after the simple `scale_random_pairs` code.

To execute, provide the number of coins to test on the command line and look at lines that begin with the string "Unfair":

```
$ python coins.py 4 | grep "^Unfair"
Unfair coin found after 3 steps: (1H, 1)
Unfair coin found after 3 steps: (1L, -1)
Unfair coin found after 3 steps: (2H, 1)
Unfair coin found after 2 steps: (2L, -1)
Unfair coin found after 3 steps: (3H, 1)
Unfair coin found after 3 steps: (3L, -1)
Unfair coin found after 3 steps: (4H, 1)
Unfair coin found after 3 steps: (4L, -1)
```

The trials data is provided in the `trials.tar.gz` archive, which includes step-by-step instructions (README) for recreating them and the figure in `gnuplot`.