Counterfeit Coin – AI miniproject, Fall 2008

Hao Du (Dec.8, 2008)

Introduction

The `Counterfeit Coin Measurement' is a very interesting, well-defined problem. I wrote a program to solve for the Solution-Tree. The program consider partially constructed Solution-Trees as states and use Depth-First-Search. The program accepts the following three arguments as input,

CoinN :	a variable number of coins
MeasureN :	a variable number of maximum measurements
Heuristic :	using the heuristic strategy?

- The `coinsolver.cpp' was done with Visual Studio 2008. There are no additional dependences required, compile and run it will be fine.
- The `results' folder contains sample Solution-Trees stored in human-readable TXT files.

A Strategy for Reducing the Branching factor

The program applies the following strategy to reduce the branching factor. Given a node in the solution-tree, instead of considering all possible compositions of coins to measure, it first classify the coins into four categories according to the belief information at that node, (1) normal coin; (2) only possible lighter coin; (3) only possible heavier coin; (4) possible both lighter and heavier coin. Coins in the same category are treated equally.

This greatly reduces the branching factor. To give an example, for CoinN=12 at the root node, there are initially C(12,6)*C(6,6)+C(12,5)*C(7,5)+C(12,4)*C(8,4)+C(12,3)*C(9,3)+ C(12,2)*C(10,2)+C(12,1)*C(11*1) which is thousands of possible measurements. But since all coins there belong to category (4), there are only 6 possible measurements that are inherently different with each other, (1vs1), (1 2 vs 3 4), (1 2 3 vs 4 5 6), (1 2 3 4 vs 5 6 7 8), (1 2 3 4 5 vs 6 7 8 9 10) and (1 2 3 4 5 6 vs 7 8 9 10 11 12).

RESULTS

The following table lists runtime results (Number of solution-trees explored, Time-cost) given different inputs under `non-heuristic' and `heuristic' cases. The software and hardware environment is `Vmware virtual WindowsXP, MacBook with CPU of 2.4GHz Intel Core 2 Duo'.

CoinN	MeasureN	Solution Found?	Run Time Results	
			Non Heuristic	Heuristic
2	2	NO	3 trees, 0.0 secs	3 trees, 0.0 secs
3	2	YES	15 trees, 0.0 secs	13 trees, 0.0 secs
4	2	NO	35 trees, 0.0 secs	35 trees, 0.0 secs
4	3	YES	44 trees, 0.0 secs	40 trees, 0.0 secs
6	3	YES	111 trees, 0.0 secs	40 trees, 0.0 secs
8	3	YES	2058 trees, 0.0 secs	40 trees, 0.0 secs
10	3	YES	31125 trees, 0.4 secs	40 trees, 0.0 secs
11	3	YES	89469 trees, 1.1 secs	40 trees, 0.0 secs
12	3	YES	146228 trees, 2.7 secs	40 trees, 0.0 secs
13	3	NO	268530 trees, 5.7 secs	268530 trees, 11.6 secs
13	4	YES	203288 trees, 3.6 secs	121 trees, 0.0 secs
14	4	YES	323461 trees, 8.3 secs	121 trees, 0.0 secs
15	4	YES	708305 trees, 19.2 secs	121 trees, 0.0 secs
16	4	YES	12934160 trees, 487secs	121 trees, 0.0 secs
20	4	YES	too big	121 trees, 0.1 secs
30	4	YES	too big	121 trees, 0.2 secs
39	4	YES	too big	121 trees, 0.4 secs
40	5	YES	too big	364 trees, 0.5 secs
50	5	YES	too big	364 trees, 1.6 secs
55	6	YES	too big	1093 trees, 2 secs
…				

Here is the Solution-Tree to the {CoinN=12, MeasureN=3} Problem. What is interesting here is that, we can have an arbitrary counterfeit coin in mind, and quickly go through the following solution-tree(text) to see if the the tree can figure it out.

```
Node 0   ( 1 2 3 4 .vs. 5 6 7 8 )
         Goto [1 | 14 | 27] for [left_heavy | equal | right_heavy]
Node 1   ( 5 6 1 2 .vs. 9 10 11 7 )
         Goto [2 | 6 | 10] for [left_heavy | equal | right_heavy]
Node 2   ( 7 1 .vs. 3 4 )
         Goto [3 | 4 | 5] for [left_heavy | equal | right_heavy]
Node 3   Ans = 1 heavy
Node 4   Ans = 2 heavy
Node 5   Ans = 7 light
Node 6   ( 8 3 .vs. 1 2 )
         Goto [7 | 8 | 9] for [left_heavy | equal | right_heavy]
Node 7   Ans = 3 heavy
Node 8   Ans = 4 heavy
```

Node 9 Ans = 8 light
Node 10 (5 .vs. 1)
Goto [11 | 12 | 13] for [left_heavy | equal | right_heavy]
Node 11 Impossible
Node 12 Ans = 6 light
Node 13 Ans = 5 light
Node 14 (9 10 11 .vs. 1 2 3)
Goto [15 | 19 | 23] for [left_heavy | equal | right_heavy]
Node 15 (9 .vs. 10)
Goto [16 | 17 | 18] for [left_heavy | equal | right_heavy]
Node 16 Ans = 9 heavy
Node 17 Ans = 11 heavy
Node 18 Ans = 10 heavy
Node 19 (12 .vs. 1)
Goto [20 | 21 | 22] for [left_heavy | equal | right_heavy]
Node 20 Ans = 12 heavy
Node 21 Impossible
Node 22 Ans = 12 light
Node 23 (9 .vs. 10)
Goto [24 | 25 | 26] for [left_heavy | equal | right_heavy]
Node 24 Ans = 10 light
Node 25 Ans = 11 light
Node 26 Ans = 9 light
Node 27 (1 2 5 6 .vs. 9 10 11 3)
Goto [28 | 32 | 36] for [left_heavy | equal | right_heavy]
Node 28 (3 5 .vs. 1 2)
Goto [29 | 30 | 31] for [left_heavy | equal | right_heavy]
Node 29 Ans = 5 heavy
Node 30 Ans = 6 heavy
Node 31 Ans = 3 light
Node 32 (4 7 .vs. 1 2)
Goto [33 | 34 | 35] for [left_heavy | equal | right_heavy]
Node 33 Ans = 7 heavy
Node 34 Ans = 8 heavy
Node 35 Ans = 4 light
Node 36 (1 .vs. 3)
Goto [37 | 38 | 39] for [left_heavy | equal | right_heavy]
Node 37 Impossible
Node 38 Ans = 2 light
Node 39 Ans = 1 light

For more Solution-Trees like {CoinN=55,MeasureN=5}, etc, please see in the `results' folder.