Mini-Project: Solving the 12 Coins Problem Using Search

Emily Wang

CSE 573
December 11, 2008

1 Introduction

The 12 Coins Problem is stated in a more general form as follows. Given $c = (3^n - 3)/2$ coins for some $n > 1$, and the fact that $c - 1$ of them are identical and the single counterfeit coin can be told apart by its differing weight, formulate a plan for identifying the counterfeit coin and whether it is heavier or lighter, by making only n measurements using a balancing scale.[1]

2 Implementation

A search algorithm for solving this puzzle for an arbitrary number of coins was implemented in Common Lisp. The program takes a list of named coins (e.g., `'(coin01 coin02 ... coin12)`) as input and searches for a set of no more than $\lceil \log_3 (2c + 3) \rceil$ measurements for an input list of length c, which is guaranteed to find the identity (a coin name) and the characteristic (either heavier or lighter) of the counterfeit coin.

CLOS (the Common Lisp Object System) was used to define a compact representation of the planning aspect and the search aspect of the solution, as well as for overloading names of functions that specialize on particular heuristics.

[1] `http://www.iwriteiam.nl/Ha12coins.html`

2.1 Plan Representation

The planned set of balance measurements is represented in the form of a decision tree with a uniform branching factor of 3. Each non-leaf node represents a measurement, with three children that correspond to the three possible outcomes: coins on the left-side weigh more, the scale balances, or coins on the right-side weigh more.

Each node contains a `belief-state` object, which has two slots: `heavy` is a list of coins that could be overweight, and `light` is a list of coins that could be underweight. Both slots are initialized to the full input list of named coins. In the method `reduce-beliefs`, coins are removed from `heavy` when their side of the scale weighed less, and vice-versa for `light`. When the scale balances, coins that participated in the measurement on either side of the scale are eliminated from both `heavy` and `light`.

A plan that satisfies the goal has $2c$ leaves where only one coin remains in each `belief-state`, and each coin appears in `heavy` of some belief, and `light` of some belief, both exactly once. Any other leaf nodes have an empty `belief-state`, which indicates the path to that leaf involves a set of incongruent (impossible) measurement observations.

2.2 Search Representation

Since plans are only valid if they perform n or less measurements, we only want to search within the space of decision trees of depth up to $n + 1$, when counting the root node. Thus, depth-limited search is a natural choice for an uninformed search strategy.

The search algorithm also only considers measurements that place (approximately) one-third of the coins on each side of the balance, leaving the remaining third out. (This is approximate in the case of c that is not perfectly divisible by three.) This assumption cannot reduce the information gained by the measurement, because we are assured that one and only one coin is counterfeit, and so adding an equal number of genuine coins to either side of the balance does not change the outcome of the measurement.

Therefore, each measurement is encoded as a set of three disjoint subsets of the coins, of size $\lfloor c/3 \rfloor$: `group0` for the left-hand pan of the scale, `group1` for the right-hand pan of the scale, and `group2` that does not take part directly in the measurement (although information about it can be inferred). To enumerate all possible combinations of disjoint subsets, utility functions were taken from third-party code in *L-99: Ninety-Nine Lisp Problems.*[2] For $c = 12$ there are 675 such combinations, which is the effective branching factor for the uninformed search. (Permutations within subsets are not counted, but the same three disjoint subsets in a different order do count as separate combinations.)

The heuristic search version of the algorithm estimates the number of additional measurements needed given the `belief-state` of a node by $\log_3$ of the size of the belief, and then conducts a greedy search, preferentially expanding nodes with a small `belief-state`. If the cardinality of a belief is so great that it would require more than $n-$ the current depth more measurements to resolve the puzzle, the search node is not pushed back onto the list of candidates for expansion. Since the total path cost of the solution is known and known to be achievable, A* search is no more appropriate for this domain then greedy search.

Heuristics are specified using subclasses (e.g., `h2-search-node`), which also inherit slots from the `search-node` superclass. The heuristics themselves are encoded in the behavior of methods (like functions) that specialize on particular subclasses of `search-node`.

Time permitting, a `navigate-plan` function could be implemented to take a completely specified measurement plan, and output a path from the root to a leaf node according to user-specified measurement outcomes. However the same functionality can be achieved using the Inspector, if running Lisp in Emacs with Slime.

2.3 Example Usage

The following REPL inputs will run the demonstration examples.[3]

```
CL-USER> (load "search.lisp")
; Loading /Users/wang/Desktop/coins/search.lisp
;   Loading /Users/wang/Desktop/coins/p27.lisp
```

[2] `http://www.ic.unicamp.br/~meidanis/courses/mc336/2006s2/funcional/L-99_Ninety-Nine_Lisp_Problems.html`

[3] "File not found" errors can usually be fixed by changing Lisp's current working directory. In the REPL, this is done by typing, ",cd [dir]." In an Emacs buffer while using SLIME, the command "C-c ~" changes the working directory to the directory of the buffer's file.

```
;     Loading /Users/wang/Desktop/coins/p07.lisp
;     Loading /Users/wang/Desktop/coins/p26.lisp
;       Loading /Users/wang/Desktop/coins/p17.lisp
;   Loading /Users/wang/Desktop/coins/coins.lisp
;     Loading /Users/wang/Desktop/coins/h2-coins.lisp
T

CL-USER> (do-search *uninformed-12coins*)
#<PLAN-TREE @ #x106ce4f2>    ; right-click to view in Inspector

CL-USER> (do-search *heuristic-12coins*)
#<PLAN-TREE @ #x133c190a>    ; right-click to view in Inspector
```

---

3 Experiment

Direct run time comparisons were not possible, because the size of the uninformed search exceeded the memory allocated by the free version of Allegro CL.[4] This occurred after running the program for about 2m.

By comparison, the heuristic greedy search for $c = 12$ executed in about 1m20s.

4 Conclusion

As a logic puzzle, the difficulty of the 12 Coins Problem stems from the large number of possible measurements. Heuristics can be used to eliminate measurements that are legal but uninformative, and also to provide a preference ordering based on the expected information gain of the remaining possible measurements. Such heuristic methods greatly decrease the time and space required to search for a completely specified measurement plan that covers all the possible cases of counterfeit coins.

[4]Hopefully this isn't a cause for taking points away(!) – it just goes to show that the heuristic search is manageable, whereas the uninformed search is very inefficient. The error message was, "An explicit gc call caused tenuring and a need for 262144 more bytes of heap. This request cannot be satisfied because you have hit the Allegro CL Free Express heap limit."