Anna Burago

Project Report. CSE 573. Fall 2008.

For my project, I have decided to implement option 1 - a counterfeit coin problem.

Implementation details:

My search space is implemented as a tree of states (class *SpaceState*). My search algorithm performs a depth-first search with backtracking (*SpaceState::SearchDepthFirstBacktrack method*) starting from a root node (*SpaceState ::CreateRootState*) till a goal state is found (*State::TestForGoalState*).

My individual search states are wrapper objects around partually constructed branching plans. Wrapper objects are implemented through class *State*. Partially constructed branching plans are implemented through class *WeightsTree*. *State* class is responsible for selecting a next node in a serach tree of states. So possible optimizations and heuristics should be implemented within this class.

Partially constructed branching plans (objects of type *WeightsTree*) are implemented as follows: each partial branching plan corresponds to a full-sized branching plan tree. On this full-sized tree nodes that correspond to current partial branching plan are specially marked. A full branching tree is a tree of depth 4, with branching level 3 at each vertice (correspond to left cup heavier, cups equal, and right cup heavier states). The branching tree is optimized for depth-first traversal since the sooner we reach to a failed leaf node the faster we can reject a current partial branching plan. So the actual nodes of the branching tree are stored in the array with the natural order corresponding to depth-first traversal. A few helper arrays within the *WeightsTree* class (*m_parentIdxMap, m_leafNodeFlags, m_ScaleObservation*)ease navigating through the tree.

The full branching tree has total of *1+3+9+27=40* nodes. Each node on a branching tree is of type *BranchTreeNode*. An object of this type stores a few pieces of information. *WeightTreeNode::m_heavyFakes* and *WeightTreeNode::m_lightFakes* arrays define a fake coin believe state that corresponds to this node. In this array an element *[i]* with the value 1 indicates a potential fake (light or heavy) coin on *i*-th place. A successfull node has only one element in two arrays equal to *1*. This array is trimmed after each observation as we move down the branching tree (function *WeightTreeNode::UpdateFromNode*). *WeightTreeNode::m_weightDistrIdx* containt the information on which coins will be weighted at this node at current iteration. This field is important for non-leaf nodes only.

I choose the order in which coins are placed on scale cups and store the information about it in individual Branch Tree nodes through the weights distribution table (*WeightDistrTable* class). This is a BIG table that contains all possible placements of coins on left and right cups (sae number of

coins on each). Each coin placement on an individual branch tree node is just an index to a row in this array. This array has total of *Comb(12,1)*Comb(11,1)+ Comb(12,2)*Comb(10,2)+ Comb(12,3)*Comb(9,3) + Comb(12,4)*Comb(8,4) + Comb(12,5)*Comb(7,5)+ Comb(12,6)*Comb(6,6)* rows. I generate this array once in the course of program execution though base-3 addition of integer numbers (*WeightDistrTable::GenerateDistributions*() function). I should have made this array twice smaller by eliminating symmetric coin placements like ((coin1 on left, coin7 on right) and (coin7 on left, coin1 on right)), but I didn't have time to do it. The size of this array corresponds to a branching factor of a states tree, which is really big.

Space complexity of my program:

Since I do depth-first search with backtracking and since the depth of a solution on the global states tree is not more then 41 (initial state plus not mode then 40 consecutive expanding partial branching plans), then the space complexity of my program is not too high. A space state at each individual moment has memory requirements of the order of *41*sizeof(full branching tree plan).* And *sizeof (full branching tree plan)* has the order of *40*sizeof (individual branch node)= 40*25*sizeof(int).*

My Weights Distributions array consumes a lot of memory (see above), but still it is a reasonable amount of memory.

Time complexity of my program:

My program is very time-complex since, in fact, my space graph in a graph of depths 41 with a HUGE branching factor at each node. I do have some optimizations in place (example: adding nodes to partial weights state in depth-first order allows to drop unsecessull branching plans as soon as first failed leaf node is reached), but still this is an uninformed search.

Possible improvements.

I should have added some heuristics to my partial branching plans selection.

One possible option might have been: use 4 coins on each cup for first weighting, 2 or 3 coins for second weighting, 2 or 1 coins for 3rd weighting.

Current state of my program.

Currently I can not claim that my program is working as well as I can not claim that it is not working. I have started it with 12 coins 90 minutes ago, and it is still executing. My step-through with the smaller number of coins (modifiable in include.h) indicates that that depth-first with

backstrackingstate search algorithm is working properly. It also indicated that partially constructed branching trees correctly update believe states of it's nodes, and are properly rejected if they are not working. But I have had this step-through early in the mornign today – and I never had enough time to properly run the program with 12 coins. I should have, probably, implemented Bayesisan junk filter – it would have been easier ☺.