

# Homework 3

## LP and LDS

*Due: 5:00pm on 2/27/08*

### Problem 3.1

See the Chess programming assignment handout. Formulate the  $n \times m$ -Queens problem as an integer linear program.

**Solution:**

**Care of Jimmy Kaplowitz**

Let  $q_{x,y}$  be a two-valued variable that is equal to 0 if there is no queen in the cell at row  $x$  and column  $y$  of the  $n \times m$  chess board and 1 if a queen is present there.  $x$  is an integer ranging from 1 to  $n$ , and  $y$  is an integer ranging from 1 to  $m$ . Then the goal is to maximise

$$\sum_{i=1}^n \sum_{j=1}^m q_{i,j}$$

under the following constraints:

$$\begin{aligned} \sum_{i=1}^n q_{i,j} &\leq 1 \quad \text{for all } j \in \{1, \dots, m\} \\ \sum_{j=1}^m q_{i,j} &\leq 1 \quad \text{for all } i \in \{1, \dots, n\} \\ \sum_{j=1}^{\min(n-i+1, m)} q_{(i+j-1), j} &\leq 1 \quad \text{for all } i \in \{1, \dots, n\} \\ \sum_{i=1}^{\min(n, m-j+1)} q_{i, (i+j-1)} &\leq 1 \quad \text{for all } j \in \{2, \dots, m\} \\ \sum_{j=1}^{\min(n-i+1, m)} q_{(i+j-1), (m-j-1)} &\leq 1 \quad \text{for all } i \in \{1, \dots, n\} \\ \sum_{i=1}^{\min(n, m-j+1)} q_{(n-i-1), (i+j-1)} &\leq 1 \quad \text{for all } j \in \{2, \dots, m\}. \end{aligned}$$

**Problem 3.2**

Given the following knapsack problem:

$$\begin{array}{ll} \max & 8x_1 + 50x_2 + 49x_3 + 54x_4 \\ \text{s.t.} & 8x_1 + 10x_2 + 10x_3 + 12x_4 \leq 30 \end{array}$$

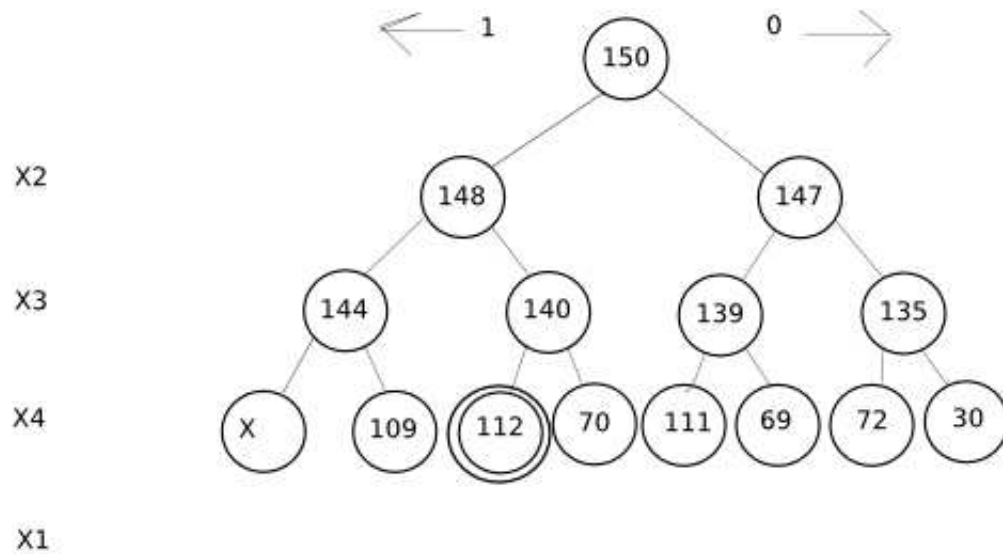
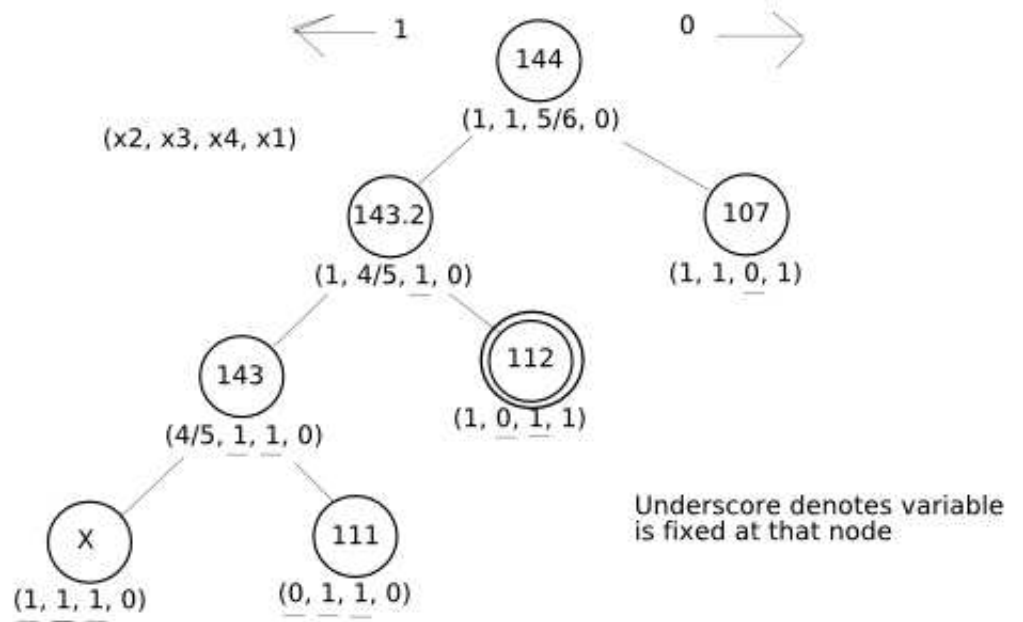
Perform two complete Least Discrepancy Searches (also called Limited Discrepancy Searches), one for each of the conditions below. Before you begin, please see the additional handout on LDS available online.

- a) The branching variable chosen is the fractional variable in the linear relaxation. As the heuristic, choose the child with the highest relaxation value. (Recall that the heuristic tells you which value to try first for a chosen branching variable.)
- b) The branching variable is the most efficient of the remaining variables that haven't been assigned a value, where efficiency is value divided by weight. Calculate the 'relaxation value' in a different way: find the most efficient unassigned item and calculate what the value would be if the knapsack were entirely filled with that item. For example, if items have efficiency (value / weight)  $4/3$ ,  $4/2$  and  $1/2$  and the capacity is 5 then the relaxation value would be  $5 \cdot 4/2 = 10$ . (This will give you an upper bound on the possible value) As the heuristic, choose the child with the greatest relaxation value, as calculated in this way.

Your handin for each problem should consist of a tree where each node contains

- a. Which variables or fractions of variables are selected
- b. The total value of the current selection (where value is calculated as per the instructions for (a) and (b))

This can either be done by hand, or with a program like Dia, but be sure the end result is readable and clear. If your tree is getting unreasonably large, you are probably doing something wrong.

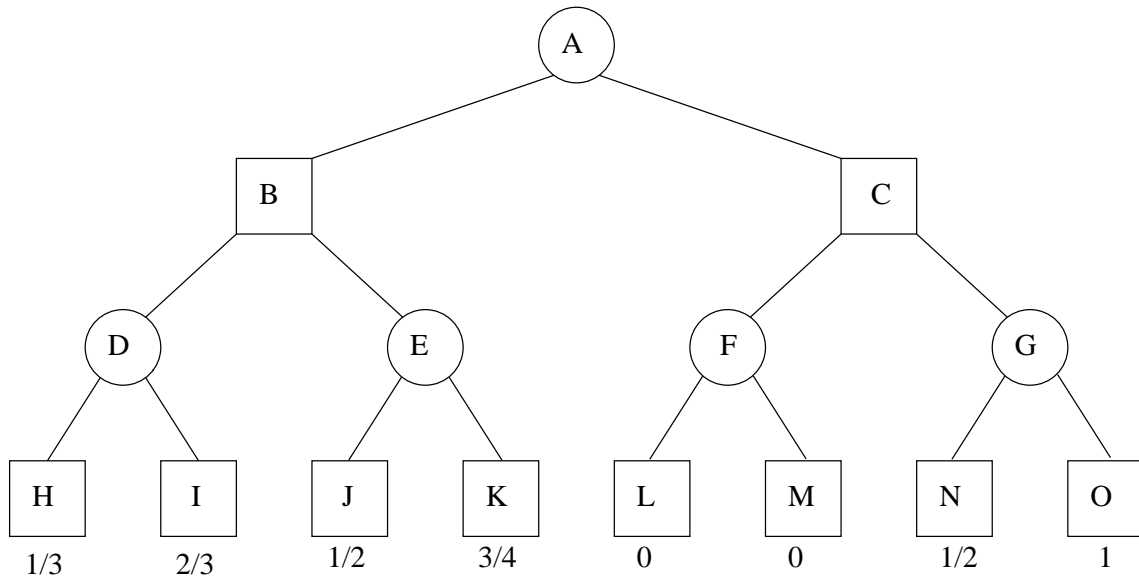


Solution:

**Problem 3.3**

Run **Alpha-Beta-Search** (page 170 in Russel and Norvig) on the game tree given in figure. Numbers below a node are the result of calling **Utility** on that node.

- a. Fill in the table with the values of  $\alpha$  and  $\beta$  when **Max-Value** or **Min-Value** is called on each node.
- b. Put a slash through any branch that is pruned: *e.g.*, if the search visits node  $X$  but does not visit its child node  $Y$ , put a slash through the edge connecting  $X$  to  $Y$ .



Node	$\alpha$	$\beta$
A		
B		
C		
D		
E		
F		
G		
H		
I		
J		
K		
L		
M		
N		
O		

Solution:

This solution uses the recursive  $\alpha\beta$ Pruning algorithm given on page 12 of Amy's notes on Adversarial Search:

Node	$\alpha$	$\beta$
A	$-\infty$	$\infty$
B	$-\infty$	$\infty$
C	2/3	$\infty$
D	$-\infty$	$\infty$
E	$-\infty$	2/3
F	2/3	$\infty$
G	—	—
H	$-\infty$	$\infty$
I	1/3	$\infty$
J	$-\infty$	2/3
K	1/2	2/3
L	2/3	$\infty$
M	2/3	$\infty$
N	—	—
O	—	—

—: This node is pruned.

### Problem 3.4

Suppose you are given a two-player, zero-sum game  $G = \langle \{\text{Max}, \text{Min}\}, X, S, T, \delta, l, v \rangle$ . To find the minimax value of the game, while expanding as few nodes as possible, you would probably use  $\alpha\beta$ Pruning over Minimax.

Now suppose you also have a good, but not perfect, static evaluation function  $e : X \rightarrow \mathbb{R}$  (a static evaluation function takes a game state and returns a guess based on that state, returns a number that is a good guess as to the value of that state). Design an algorithm that is expected to perform better (expand fewer nodes).

Specifically:

- Your algorithm should return the exact minimax value, not an approximation.
- You may assume that  $S$ , the set of start states, is a singleton, with distinct element  $s$ .

- You are not given anything beyond  $G$  and  $e$ . If you design an algorithm, it must work given only those two inputs.
- You may assume that applying the static evaluation function to a node does not use excessive time or space.
- Speed is not the primary concern (the number of visited nodes is), but algorithms that are needlessly slow will be penalized.
- An algorithm does not need to visit strictly fewer nodes in all cases to be considered an improvement: *e.g.*,  $\alpha\beta$ Pruning does not visit strictly fewer nodes than MiniMax in all cases, but it is still considered better than Minimax because it never visits more nodes and it sometimes visits strictly fewer nodes.

You may present your answer in pseudo-code or you may write a clear explanation. You should also provide a brief explanation of why your algorithm performs better than  $\alpha\beta$ Pruning.

**Solution:** You can use  $e$  to sort the nodes before each recursive call of  $\alpha\beta$ Pruning. At Maxie nodes one would order them from largest to smallest and at Minnie from smallest to largest. Initially, the most promising path would be explored. Even if this path does lead to a minimax solution, hopefully many of the less promising children could be pruned.