6.034 Quiz 1, Spring 2005

1 Search Algorithms (16 points)

1.1 Games

The standard alpha-beta algorithm performs a depth-first exploration (to a pre-specified depth) of the game tree.

1. Can alpha-beta be generalized to do a breadth-first exploration of the game tree and still get the optimal answer? Explain how or why not. If it can be generalized, indicate any advantages or disadvantages of using breadth-first search in this application.

   No. The alpha-beta algorithm is an optimization on min-max. Min-max inherently needs to look at the game-tree nodes below the current node (down to some pre-determined depth) in order to assign a value to that node. A breadth-first version of min-max does not make much sense. Thinking about alpha-beta instead of min-max only makes it worse, since the whole point of alpha-beta is to use min-max values from one of the earlier (left-most) sub-trees to decide that we do not need to explore some later (right-most) subtrees.

   Some answers suggested that min-max inherently needs to go all the way down to the leaves of the game tree, where the outcome of the game is known. This is not true. Typically one picks some depth of look-ahead depth and searches to that depth, using the static evaluator to compute a score for the board position at that depth.

2. Can alpha-beta be generalized to do a progressive-deepening exploration of the game tree and still get the optimal answer? Explain how or why not. If it can be generalized, indicate any advantages or disadvantages of using progressive-deepening search in this application.

   Yes. Progressive-deepening involves repeated depth-first searches to increasing depths. This can be done trivially with min-max and alpha-beta as well, which also involve picking a maximum depth of lookahead in the tree. PD does waste some work, but as we saw in the notes, the extra work is a small fraction of the work that you would do anyways, especially when the branching factor is high, as it is in game trees. The advantage is that in timed situations you guarantee that you always have a reasonable move available.

1.2 Algorithms

1. You are faced with a path search problem with a very large branching factor, but where the answers always involve a relative short sequence of actions (whose exact length is unknown). All the actions have the same cost. What search algorithm would you use to find the optimal answer? Indicate under what conditions, if any, a visited or expanded list would be a good idea.

   Progressive deepening (PD), with no visited or expanded list would probably be the best choice. All the costs are the same, so breadth-first (BF) and PD both guarantee finding the shortest path in that situation, without the overhead of uniform-cost search. Since the branching factor is high, space will be an issue, which is why we prefer PD over BF. If we were to use a visited list with PD, the space cost would be the same as BF and it would not make sense to pay the additional run-time cost of PD (repeated exploration of parts of the tree) if we give up the space advantage.

2. You are faced with a path search problem with a very large branching factor, but where the answers always involve a relative short sequence of actions (whose exact length is unknown). These actions, however, have widely varying costs. What search algorithm would you use to find the optimal answer? Indicate under what conditions, if any, a visited or expanded list would be a good idea.

   Since we have variable link costs, we should use Uniform Cost search to guarantee the optimal answer. The fact that the costs are highly variable is good, since we expect that we might be able to avoid exploring sub-trees with high cost. Note that we don't necessarily have a useful heuristic and so A* may not be applicable. Using an expanded list would make sense if the search space involves lots of loops, which would lead us to re-visit the same state many times. However, since we know that there's a relatively short path to the goal, it might not be worth the extra space.

2 Constraints (16 points)

Consider assigning colors to a checkerboard so that squares that are adjacent vertically or horizontally do not have the same color. We know that this can be done with only two colors, say red (R) and black (B). We will limit our discussion to **five squares** on a 3x3 board, numbered as follows:

```
1 | 2 | 3
---------
4 | 5 |
---------
  |   |
```

Let's look at the CSP formulation of this problem. Let the squares be the variables and the colors be the values. All the variables have domains { R, B }.

1. If we run full constraint propagation on the initial state, what are the resulting domains of the variables?

   None of the variable domains change:

   $$1 = \{R, B\} \qquad 2 = \{R, B\} \qquad 3 = \{R, B\}$$
   $$4 = \{R, B\} \qquad 5 = \{R, B\}$$

2. Say, instead, the initial domain of variable 5 is restricted to { B }, with the other domains as before. If we now run full constraint propagation, what are the resulting domains of the variables?

   $$1 = \{B\} \qquad 2 = \{R\} \qquad 3 = \{B\}$$
   $$4 = \{R\} \qquad 5 = \{B\}$$

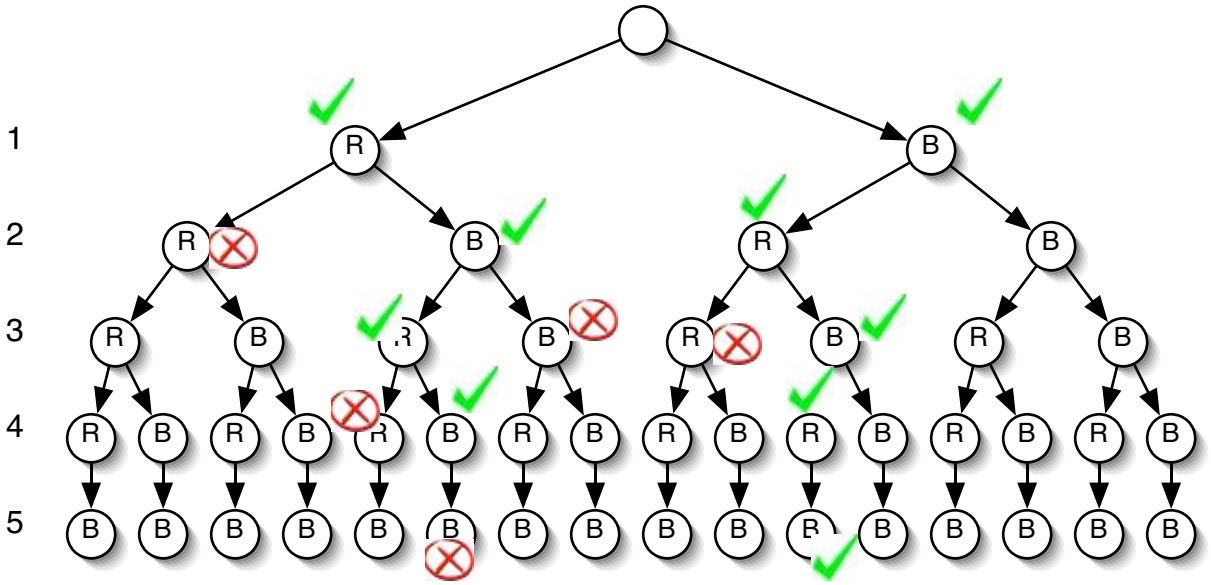
3. If in the initial state (all variables have domains { R, B }), we assign variable 1 to R and do forward checking, what are the resulting domains of the other variables?

   Forward checking is defined as a single iteration of constraint propagation only on those edges that terminate at the variable whose value was just set, and that do not originate from variables which have already been set. Therefore, after we set $1 = R$, forward checking affects the domains of variables 2 and 4 since they are adjacent to variable 1 (and have not yet been assigned).

   $$1 = \{R\} \qquad 2 = \{B\} \qquad 3 = \{R, B\}$$
   $$4 = \{B\} \qquad 5 = \{R, B\}$$

   Forward checking only does one step of propagation, only to the immediate neighbors of the assigned variable.

4. Assume that during backtracking we first attempt assigning variables to R and then to B. Assume, also, that we examine the variables in numerical order, starting with 1. Also, let the domain of variable 5 be { B }, the other domains are { R, B }. In the following tree, which shows the space of assignments to the 5 variables we care about, indicate how pure backtracking (BT) would proceed by placing a check mark next to any assignment that would be attempted during the search and crossing out the nodes where a constraint test would fail. Leave unmarked those nodes that would never be explored.



5. If we use backtracking with forward checking (BT-FC) in this same situation, give a list of all the assignments attempted, in sequence. Use the notation variable = color for assignments, for example, 1=R.

We must keep track of the variable domains as we search since forward checking modifies these domains based on the current assignment, and we will need to restore the domain of earlier search nodes if we have to backtrack to them. We fail at a node if (1) the current assignments violate some constraint, or (2) if forward checking after the present assignment causes the domain of some variable to become empty. The following lists (in order from left to right) each attempted assignments and the resulting variable domains *after* forward checking.

Assignment:	None	$1 = R$	$2 = B$	$1 = B$	$2 = R$	$3 = B$	$4 = R$	$5 = B$
Domain of 1:	$\{R, B\}$	**R**	**R**	**B**	**B**	**B**	**B**	**B**
Domain of 2:	$\{R, B\}$	$\{B\}$	**B**	$\{R\}$	**R**	**R**	**R**	**R**
Domain of 3:	$\{R, B\}$	$\{R, B\}$	$\{R\}$	$\{R, B\}$	$\{B\}$	**B**	**B**	**B**
Domain of 4:	$\{R, B\}$	$\{B\}$	$\{B\}$	$\{R\}$	$\{R\}$	$\{R\}$	**R**	**R**
Domain of 5:	$\{B\}$	$\{B\}$	$\{\}$	$\{B\}$	$\{B\}$	$\{B\}$	$\{B\}$	**B**
			$\Downarrow$					
			FAIL					

Note that when we fail at $2 = B$, since there are no further values to try in the

domain of variable 2, we backtrack to the assignment of variable 1. When this happens, we restore the domains from *before* variable 1 was assigned, i.e. the ones listed above under "None".

6. If we use backtracking with forward checking (BT-FC) but with dynamic variable ordering, using the most-constrained-variable strategy, give a list of all the variable assignments attempted, in sequence. If there is a tie between variables, use the lowest-numbered one first. Use the notation variable = color for assignments, for example, 1=R.

   Use of the most-constrained-variable strategy entails assigning the variable first whose domain is smallest. This ordering is not only performed at the start of the search. Rather, it is updated after each variable is assigned and forward checking modifies the domains of unassigned variables. For this problem, variable 5 has the smallest domain initially. After assigning variable 5, the domains of variable 2 and 4 become smaller than those of variables 3 and 5. Since variable 2 has the lowest index, it is assigned next. And so on, as shown below:

Assignment:	None	$5 = B$	$2 = R$	$1 = B$	$3 = B$	$4 = R$
Domain of 1:	$\{R, B\}$	$\{R, B\}$	$\{B\}$	**B**	**B**	**B**
Domain of 2:	$\{R, B\}$	$\{R\}$	**R**	**R**	**R**	**R**
Domain of 3:	$\{R, B\}$	$\{R, B\}$	$\{B\}$	$\{B\}$	**B**	**B**
Domain of 4:	$\{R, B\}$	$\{R\}$	$\{R\}$	$\{R\}$	$\{R\}$	**R**
Domain of 5:	$\{B\}$	**B**	**B**	**B**	**B**	**B**

3 Constraint satisfaction (24 points)

You are trying to schedule observations on the space telescope. We have m scientists who have each submitted a list of n telescope observations they would like to make. An observation is specified by a target, a telescope instrument, and a time slot. Each scientist is working on a different project so the targets in each scientist's observations are different from those of other scientists. There are k total time slots, and the telescope has three instruments, but all must be aimed at the same target at the same time.

The greedy scientists cannot all be satisfied, so we will try to find a schedule that satisfies the following constraints:

C1. Exactly two observations from each scientist's list will be made (the choice of the two will be part of the solution).

C2. At most one observation per instrument per time slot is scheduled.

C3. The observations scheduled for a single time slot must have the same target.

Note that for some set of requested observations, there may not be any consistent schedule, but that's fine.
Consider the following three formulations of the problem.

A. The variables are the $3k$ instrument/time slots.

B. The variables are the m scientists.

C. The variables are the mn scientists' requests.

For each formulation, specify

1. The value domain for the variables.

2. The size of the domain for the variables (in terms of k, m, and n).

3. Which of the constraints are necessarily satisfied because of the formulation.

4. Whether the constraints can be specified as binary constraints in this formulation. If they can, explain how. If not, provide a counterexample.

Formulation A: The variables are the $3k$ instrument/time slots.
1. Domain: **for each instrument/time slot, the set of observations requesting that instrument and time slot and the value "empty"**
2. Size of domain: **at most m*n+1 per variable**
3. Satisfied constraints: **C2, since each variable (instrument/time) gets at most one value, an observation.**
4. Binary constraints?:

- **C1 is not a binary constraint in this formulation. It requires checking all the variable assignments at once to make sure that exactly two observations from each scientist's list are made.**
- **C3 is a binary constraint in this formulation. Place a constraint between the 3 variables with the same time slot and require that the targets of the assigned observation be equal if they are both non-empty.**

Formulation B: The variables are the m scientists.
1. Domain: **for each scientist, the set of all pairs of observations that scientist requested.**
2. Size of domain: $\binom{n}{2}$**, approximately** $n^2/2$**.**
3. Satisfied constraints: **C1, since we will guarantee that exactly two of the scientist's observations are scheduled.**
4. Binary constraints?:

- **C2 is a binary constraint in this formulation. Place a constraint between every pair of variables and require that the instrument/time slot requests don't conflict.**
- **C3 is a binary constraint in this formulation. Place a constraint between every pair of variables and require that the targets for observations with the same time slot don't conflict.**

Formulation C: The variables are the mn scientists' requests.
1. Domain: {Granted, Rejected}
2. Size of domain: 2
3. Satisfied constraints: None
4. Binary constraints?:

- **C1 is not a binary constraint in this formulation. It requires checking all the variable assignments of Granted observations at once to make sure that exactly two observations from each scientist's list are granted.**
- **C2 is a binary constraint in this formulation. Place a constraint between every pair of variables and require that the instrument/time slot requests don't conflict between any two Granted requests.**
- **C3 is a binary constraint in this formulation. Place a constraint between every pair of variables and require that the targets of the Granted observations with the same time slot don't conflict.**

4 Search Problem formulation (23 points)

Consider a Mars rover that has to drive around the surface, collect rock samples, and return to the lander. We want to construct a plan for its exploration.

- It has batteries. The batteries can be charged by stopping and unfurling the solar collectors (pretend it's always daylight). One hour of solar collection results in one unit of battery charge. The batteries can hold a total of 10 units of charge.
- It can drive. It has a map at 10-meter resolution indicating how many units of battery charge and how much time (in hours) will be required to reach a suitable rock in each square.
- It can pick up a rock. This requires one unit of battery charge. The robot has a map at 10-meter resolution that indicates the type of rock expected in that location and the expected weight of rocks in that location. Assume only one type of rock and one size can be found in each square.

The objective for the rover is to get one of each of 10 types of rocks, within three days, while minimizing a combination of their total weight and the distance traveled. You are given a tradeoff parameter α that converts units of weight to units of distance. The rover starts at the lander with a full battery and must return to the lander.

Here is a list of variables that might be used to describe the rover's world:

- types of rocks already collected
- current rover location (square on map)
- current lander location (square on map)
- weight of rocks at current location (square on map)
- cost to traverse the current location (square on map)
- time since last charged
- time since departure from lander
- current day
- current battery charge level
- total battery capacity
- distance to lander
- total weight of currently collected rocks

1. Use a set of the variables above to describe the rover's state. Do not include extraneous information.

   - **types of rocks already collected**
   - **current rover location (square on map)**
   - **time since departure from lander**
   - **current battery charge level**
   - **total weight of currently collected rocks (optional, depending on your choice of cost function)**

2. Specify the goal test.

   - **All types of rocks have been collected**
   - **rover at lander location**
   - **time since departure less than 3 days**

3. Specify the actions. Indicate how they modify the state and any preconditions for being used.

   charge **: precondition: none; effects: increases battery voltage by 1 unit, increases time-since-departure by 1 hour**

   move **: precondition: enough battery voltage to cross square; effects: decreases battery voltage by amount specified in map; increases time by amount specified in map; changes rover location**

   pick-up-rock **: precondition: enough battery voltage; effects: decreases battery voltage by 1 unit; changes types of rocks already collected**

4. Specify a function that determines the cost of each action.

   charge **: 0**

   move **: 10 meters**

   pick-up-rock **:** α *** weight-of-rocks-at-current-location**

5. This can be treated as a path search problem. We would like to find a heuristic. Say whether each of these possible heuristics would be useful in finding the optimal path or, if not, what's wrong with them. Let l be the number of rocks already collected.

   H1: The sum of the distances (in the map) from the rover to the $10 - l$ closest locations for the missing types of rocks.

   This heuristic is inadmissible.

   H2: The length of the shortest tour through the $10 - l$ closest locations for the missing types of rocks.

   This heuristic would take an impractical amount of time to compute; and while more reasonable than H1 is also inadmissible.

   H3: The distance back to the lander.

   This heuristic is admissible, but very weak.

5 Search traces (21 points)

Consider the graph shown in the figure below. We can search it with a variety of different algorithms, resulting in different search trees. Each of the trees (labeled G1 though G7) was generated by searching this graph, but with a different algorithm. Assume that children of a node are visited in alphabetical order. Each tree shows all the nodes that have been visited. Numbers next to nodes indicate the relevant "score" used by the algorithm for those nodes.

For each tree, indicate whether it was generated with

1. Depth first search
2. Breadth first search
3. Uniform cost search
4. A* search
5. Best-first (greedy) search

In all cases a strict expanded list was used. Furthermore, if you choose an algorithm that uses a heuristic function, say whether we used

H1: heuristic 1 = $\{h(A) = 3, h(B) = 6, h(C) = 4, h(D) = 3\}$

H2: heuristic 2 = $\{h(A) = 3, h(B) = 3, h(C) = 0, h(D) = 2\}$

Also, for all algorithms, say whether the result was an optimal path (measured by sum of link costs), and if not, why not. Be specific.

Write your answers in the space provided below (not on the figure).

G1: 1. Algorithm: **Breadth First Search**
2. Heuristic (if any): **None**
3. Did it find least-cost path? If not, why? **No. Breadth first search is only guaranteed to find a path with the shortest number of links; it does not consider link cost at all.**

G2: 1. Algorithm: **Best First Search**
2. Heuristic (if any): **H1**
3. Did it find least-cost path? If not, why?
No. Best first search is not guaranteed to find an optimal path. It takes the first path to goal it finds.

G3: 1. Algorithm: **A***
2. Heuristic (if any): **H1**
3. Did it find least-cost path? If not, why? **No. A* is only guaranteed to find an optimal path when the heuristic is admissible (or consistent with a strict expanded list). H1 is neither: the heuristic value for C is not an underestimate of the optimal cost to goal.**

G4: 1. Algorithm: **Best First Search**
2. Heuristic (if any): **H2**
3. Did it find least-cost path? If not, why? **Yes. Though best first search is not guaranteed to find an optimal path, in this case it did.**

G5: 1. Algorithm: **Depth First Search**
2. Heuristic (if any): **None**
3. Did it find least-cost path? If not, why? **No. Depth first search is an any-path search; it does not consider link cost at all.**

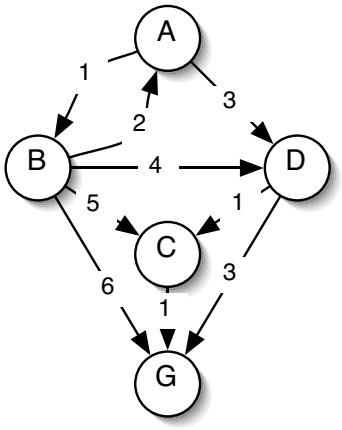
G6: 1. Algorithm: **A***
2. Heuristic (if any): **H2**

3. Did it find least-cost path? If not, why? **Yes. A* is guaranteed to find an optimal path when the heuristic is admissible (or consistent with a strict expanded list). H2 is admissible but not consistent, since the link from D to C decreases the heuristic cost by 2, which is greater than the link cost of 1. Still, the optimal path was found.**

G7: 1. Algorithm: **Uniform Cost Search**

2. Heuristic (if any): **None**

3. Did it find least-cost path? If not, why? **Yes. Uniform Cost is guaranteed to find a shortest path.**



	H1	H2
A	3	3
B	6	3
C	4	0
D	3	2