CS472 Foundations of Artificial Intelligence
Fall 2000
Assignment 1
SOLUTION

1. **Uninformed Search**

   (a) Consider a tree with 2 top-most children, one of which is a goal, the other the root of a subtree which has no goal nodes. Breadth-first search will find the goal storing at most the sibling of the goal. But if a depth-first search starts with the goal's sibling, it will have to store the current path while (fruitlessly) searching the subtree.

   There is no tree for which DFS *always* requires more memory than BFS, because DFS could always expand the correct node first.

   Common mistakes.

   - Missing the last part about the existence of a tree where DFS always requires more memory.
   - Assuming that DFS always searches from the left node to the right node, which is not true. The key word here is always!

   (b) If all we stored were the nodes along the current path, that would take $O(d)$ space. The extra storage DSF uses is for the unexplored siblings of each node on the path. Thus, we can re-generate the list of siblings in order to pick the next one when we backtrack, paying for that space savings in time. We can identify the next one as the one after the node we just backed-up from. If it is possible, and efficient, to generate a specific numbered sibling, then remembering the sibling number of each node on the path goes a long way to ameliorate that time penalty; this only adds a constant amount for space to each node on the path, so we're still $O(d)$.

   Common mistakes. Not explicitly explaining why the space requirement is reduced from $O(bd)$ to $O(m)$.

2. **Formulating Seach Problems**

   (a) i. Each state can be represented by the tuple *[stack of phone directories, position, letter]*, where *position* is the current position in the current phone book and *letter* is the last letter found in the search for the string "Juris Hartmanis". A possible initial state will be *[phone book with cities starting with A, first position in the book, the letter H]*. There are 2 types of operators. The first takes the initial state and returns a new state with the correct phone directory, namely the one containing listings for the city of Aurora, ie *[correct phone book, ?, ?]*. The second operator takes the state returned by the first operator and searches for the position in the phone book where each successively longer substring of "Hartmanis, Juris" occurs. For example, it will initially look for "H" and when found update the position and bump the letter to "a". The next seek will find "Ha", and so on. The goal state is *[Aurora phone book, position, "s" of "Juris"]*.

   ii. The initial state is the same as for part (a). Similarly, there are two operators of which the first takes the initial state and returns a new state containing the Aurora phone book. The second operator takes this state and searches the phone book for

each occurrence of the string "Juris" as a first name. The goal test is whether the phone number at the current position belongs to a Turing award winner.

iii. The state of a crossword puzzle grid is where none, some, or all of the words are filled. The initial state is the empty grid. There is one operator, which takes a state and adds a "legal" word into one of the word spaces remaining on the grid (making sure that the word doesn't conflict with any words already on the grid, ie the interecting characters must be the same). The final state is a grid that is full of legal words.

Common mistakes.

- Many people spent a lot of time describing how the search is performed, and the ways to make sure that the constraints are met, and still missed out the most important components in describing a search problem: (1) the **state representation**, i.e. what each node in your search tree represents; (2) the **initial** and **goal states** (or goal test); and (3) the **operator/s** which usually take in the current state, perform some checks and return another state.
- About half the class forgot to specify the initial state for at least one of the problems.

(b) Pretty much *anything* can be expressed as a search problem. However, if the only way to devise a goal-test is to know the answer already (i.e. simple arithmetic), then there really isn't much point to doing the search. This assumes that the problem is well-defined enough to be considered a problem w/ a solution. (Note: Any reasonably justified answer was accepted.)

Common mistakes.

- Many people listed problems using either the hardness to define the goal states and operators, or even the large size of the search space as reasons why these problems should not be search problems. However, many search problems are precisely dealing with search spaces that are exponentially big. For example, board games like Chess and GO have an infinitely large search space and the behavior of opponent players is not deterministic. Yet, these game playing problems like these have been generally regarded as search problems.
- Many like to just state down the problem without justifying why it is not a search problem.

3. **Iterative Deepening**

(a) Iterative deepining (ID) is not a good idea because it does a lot of wasteful searching. Since a completed crossword puzzle is of a known depth (i.e., a depth equal to the max. number of words the will fit in the puzzle) and there is no danger of exploring an infinite (or even very deep) path, it would make more sense to search using a simple depth-first search.

(b) Iterative broadening (IB) gives the advantage of "scanning" through the entire search space without the memory problems that breadth-first search introduces when the graph has a high branching factor. (Like iterative deepening, it also does a fair amount of wasteful searching.) Generally speaking, IB would work better than ID because it exploits the fact the goal nodes are not equally distributed among the nodes. With the extremely high branching factor induced by a large crossword dictionary, it might be possible for

IB to find a completed puzzle by pursuing only a small number of branches at each level under "good" nodes.

2 pts for correct answer, 1 if correct for wrong reason / partially correct, 0 if incorrect.

3 pts for good explaination, 2 pts for fair explaination, 1 pt for poor exlaination, 0 for no explaination.

4. **Constraint Satisfaction Problems**

(a) The most straightforward way to formalize this problem is to let each "word" be a variable whose domains are all the words of the correct length. So:

- $1A = \{ant, ape, big, bus, car, has\}$
- $1D = \{bard, book, buys, hold, lane, year, rank\}$
- $2D = \{browns, ginger, symbol, syntax\}$
- $3A = \{bard, book, buys, hold, lane, year, rank\}$
- $4A = \{ant, ape, big, bus, car, has\}$

If you like unary constraints, you could have given all the variables the same domain – all the words – and specified a unary constraint that each word has the proper number of letters to match the word size.

We have one binary constraint for each word intersection: the constraint says that the letters of the two words must match. Letting $X(n)$ denote the nth letter of word X, get get:

- $1A(1) = 1D(1)$
- $1A(3) = 2D(1)$
- $1D(3) = 3A(1)$
- $2D(3) = 3A(3)$
- $2D(5) = 4A(1)$

(b) Examining the arc $< 1A, 1D >$, we see that the words *ant, ape* and *car* can all be removed because no four letter words start with "a" or "c".

(c) The following shows which domain values are pruned under one of the many ways in which the arcs can be processed:

- $< 1A, 1D >$: ant, ape, car (from 1A)
- $< 1D, 1A >$: lane, year, rank (from 1D)
- $< 3A, 1D >$: bard, book, buys, hold
- $< 2D, 3A >$: browns, symbol
- $< 4A, 2D >$: big, bus, car, has
- $< 2D, 4A >$: syntax
- $< 1A, 2D >$: big
- $< 3A, 2D >$: year
- $< 1D, 3A >$: book, buys
- $< 1A, 1D >$: (already arc-consistent)
- $< 1D, 1A >$: (already arc-consistent)
- $< 2D, 1A >$: (already arc-consistent)

This leaves us with:

- $1A = \{bus, has\}$
- $1D = \{bard, hold\}$
- $2D = \{syntax\}$
- $3A = \{lane, rank\}$
- $4A = \{ant, ape\}$

Grading guidelines: Most students did very well on this question. Marking was flexible for part (a): full marks were given as long as answers described (or alluded to) the variables, domains, and constraints of the problems. For part (b), a few marks were deducted from a few of the answers that mentioned values that were pruned due to unary constraints, or didn't mention any concrete value or arc at all. For (c), some people's final domains weren't entirely correct – I gave 2 marks for each correct domain.

Many students tried to pose the constraint satisfaction problem as a sort of search problem (with operators, goal tests, etc.). This was unnecessary – all that's needed to describe a CSP is the variables, domains, and constraints. *Solving* a CSP often requires search, but no explicit searching was required for this question.

5. **Heuristic Search**

For all searchs, the path from the initial node is stored together with the node itself, and is parenthesized for clarity. The number in brackets represents the path cost $g(n)$ for uniform-cost search, the node heuristic $h(n)$ for best-first search, and $g(n) + h(n)$ for A* search.

(a) Uniform-cost search

1 (S)A[3] - (S)F[4] - (SF)E[6] - (SA)B[7] - (SA)F[8] - (SF)A[9] - (SFE)D[10] - (SFE)H[10] - (SAB)H[10] - (SAF)E[11] - (SFE)B[11] - (SAB)C[11] - (SAB)E[12] - (SABC)G[12]

2 SABCG

(b) Best-first search

(1) S - (S)F[6] - (SF)E[5] - (SFE)H[1] - (SFE)D[3] - (SFED)C[2] - (SFEDC)G[0]

(2) SFEDCG

(c) A^* search

(1) S - (S)A[3+7] - (S)F[4+6] - (SA)B[7+4] - (SF)E[6+5] - (SAB)H[10+1] - (SFE)H[10+1] - (SAB)C[11+2] - (SABC)G[12+0]

(2) SABCG

Grading guidelines: This was a rather cut-and-dried question, so partial credit was a little bit harder to come by. We were looking for a solid understanding of each of these algorithms, though we understand that type-os are possible. Some students seemed to have trouble following directions (many solutions broke ties in an arbitrary order instead of alphabetically). For each of the three parts:
5 points were awarded to completely correct solutions.
4 points were awarded to solutions that contained one small type-o or oversight but that demonstrated a solid understanding of the algorithm.
2 points were awarded to solutions that made large mistakes that implied a weak understanding of the algorithm.
0 points were awarded for solutions that didn't apply the search algorithm correctly at all.
Many students neglected to include G in their list of expanded nodes. Note that G must

be expanded in all of these algorithms before they terminate. Having G on the frontier is not enough (situations exist where, though G is in the set of nodes to be considered, it is not actually expanded until another non-G node is expanded first. Stop by office hours if you'd like elaboration on this point.) A one-time, 1-point penalty was imposed for question 5 solutions that repeatedly failed to expand G.