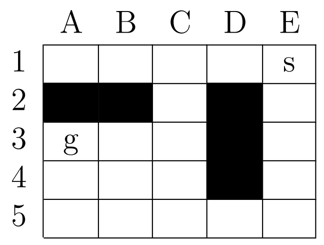
CS360 Homework 11– Solution

Breadth-First and Depth-First Search

1) A 4-neighbor gridworld is given below. In which order does depth-first search (with a sensible node pruning strategy) expand the cells when searching from s to g? Ties are broken in lexicographic order. That is, A1 is preferred over A2 and B1, and A2 is preferred over B1.



Answer:
The node pruning strategy of depth-first search is to prune nodes whenever some node on the same branch of the search tree is already labeled with the same state (cycle detection).

Expansion order: E1, D1, C1, B1, A1, (backtrack to B1 and then C1), C2, C3, B3, A3.

2) Compare the advantages and disadvantages of breadth-first and depth-first search and discuss to which degree pruning of tree nodes is important for them.

Answer:
In a finite state space, breadth-first search (BFS) and depth-first search (DFS) compare as follows:

- Both breadth-first search and depth-first search can use node pruning to decrease the number of expanded nodes and thus increase their efficiency. Breadth-first search needs to keep more information in memory than depth-first search and can then use more powerful node pruning strategies.
- BFS is complete (even without node pruning). DFS is not complete without cycle detection and thus not complete without any node pruning.
- BFS is optimal (assuming unit-cost edges), DFS offers no such guarantee.

In a finite tree with branching factor b, goal depth d, tree depth m, BFS and DFS compare as follows:

- In the worst case, BFS expands $O(b^d)$ states to find a solution, whereas DFS expands $O(b^m)$ states. Therefore, since the goal depth cannot be larger than the tree depth (that is, $d \leq m$), BFS has a better worst case performance.

- In the worst case, BFS requires memory to store $O(b^d)$ states, whereas DFS requires memory to store only $O(bm)$ states with an appropriate memory-deallocation strategy.

3) Does depth-first search always terminate if there is a path of finite length from the start to the goal? Why?

Answer:
No. If it does not use proper cycle detection, it might get stuck in a loop. Even with cycle detection, if the state space is infinite (but there is a finite length path from the start to the goal), DFS might get stuck exploring an infinite subspace of the state space.

Constraint Satisfaction

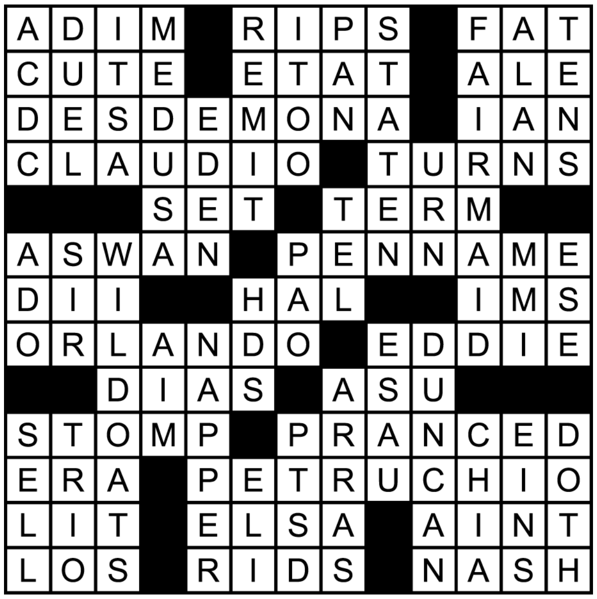
4) Consider the two formulations of the N-Queens problem as a constraint satisfaction problem from the slide set (Constraint Satisfaction, slide 9). Compare these two formulations in terms of the size and branching factor of the state space and the depth of the search tree.

Answer:
In the first formulation, there are N^2 variables, each with domain $\{0, 1\}$. Therefore, the size of the state space is 2^{N^2} (note that this is not the number of valid configurations). Since we can pick from at most two values from the domain of a variable, the branching factor is 2. Since we have to assign values to N^2 variables, the depth of the search tree is N^2.

In the second formulation, there are N variables, each with domain $\{1, \ldots, N\}$. Therefore, the size of the state space is only N^N. Since we can pick from at most N different values from the domain of a variable, the branching factor is N. Since we have to assign values to N variables, the depth of the search tree is N.

5) In the crossword puzzle, we have a grid with blocked and unblocked cells and a dictionary of words. We want to assign a letter to each unblocked cell so that each vertical or horizontal contiguous segment of unblocked cells form a word that appears in the dictionary. An example of a solved crossword puzzle is given below[1].

[1] `http://www.americanshakespearecenter.com/v.php?pg=684`



Formulate this puzzle as a constraint satisfaction problem. Describe the variables, their domains and the constraints. (Bonus question: Try to come up with a second formulation of this puzzle as a constraint satisfaction problem.)

Answer:
Formulation 1:

Variables: Unblocked cells.

Domains: The domain of each variable is the set of letters of the alphabet.

Constraints: For each vertical or horizontal contiguous segment of unblocked cells, we add a constraint between the cells in that segment, constraining the letters assigned to the cells in that segment to form a word that appears in the dictionary.

Formulation 2:

Variables: All vertical or horizontal contiguous segments of unblocked cells.

Domains: The domain of each variable is the set of words in the dictionary of the same length as the corresponding contiguous segment.

Constraints: For each pair of vertical and horizontal contiguous segments of unblocked cells that intersect at an unblocked cell s, we add a constraint between them, constraining the words assigned to them to have the same letter at s.

6) Suppose you have a search problem defined by more or less the usual stuff:

- a set of states S;
- an initial state s_{start};
- a set of actions A including the $NoOp$ action, that has no effect;

- a transition model $Result(s, a)$ (that determines the successor state when action a is executed in state s);
- a set of goal states G.

Unfortunately, you have no search algorithms! All you have is a CSP solver.

(a) Given some time horizon T, explain how to formulate a CSP such that (1) the CSP has a solution exactly when the problem has a solution of length T steps; (2) the solution to the original problem can be "read off" from the variables assigned in CSP solution. Your formulation must give the variables, their domains, and all applicable constraints expressed as precisely as possible. You should have at least one variable per time step, and the constraints should constrain the initial state, the final state, and consecutive states along the way.

(b) Explain how to modify your CSP formulation so that the CSP has a solution when the problem has a solution of length $\leq T$ steps, rather than exactly T steps.

Answer:
We use the variables $s_0 \ldots s_T$ and $a_0, \ldots a_{T-1}$. The domain of s_0 is $\{s_{start}\}$, the domain of s_T is G, the domains of $s_1, \ldots s_{T-1}$ are S, and the domains of $a_0, \ldots a_{T-1}$ are A. For each $i = 0 \ldots T-1$, we have the following constraint: $Result(s_i, a_i) = s_{i+1}$, that is, the execution of action a_i in state s_i results in state s_{i+1} (and $a_i \in A(s_i)$, where $A(s_i)$ is the set of actions that can be executed in s_i). If we include the $NoOp$ action in A (and all $A(s_i)$), we find a solution of length $\leq T$ steps. Otherwise, we find a solution of length T steps. In both cases, the solution can be read off from the assignments to the variables $a_0 \ldots a_{T-1}$.