Assignment 2

Search Problems in AI

Deadline: October 28th, 11:55pm.
Perfect score: 100.

Assignment Instructions:

Teams: Assignments should be completed by teams of up to three students. No additional credit will be given for students that complete an assignment individually. Please inform the TAs as soon as possible about the members of your team so they can update the scoring spreadsheet (find the TAs' contact info under the course's site on Sakai).

Submission Rules: Submit your reports electronically as a PDF document through Sakai (`sakai.rutgers.edu`). For programming questions, you need to also submit a compressed file via Sakai, which contains your code. Do not submit Word documents, raw text, or hardcopies etc. Make sure to generate and submit a PDF instead. Each team of students should submit only a single copy of their solutions and indicate all team members on their submission. Failure to follow these rules will result in lower grade in the assignment.

Late Submissions: No late submission is allowed. 0 points for late assignments.

Extra Credit for LaTeX: You will receive 10% extra credit points if you submit your answers as a typeset PDF (using LaTeX, in which case you should also submit electronically your source code). There will be a 5% bonus for electronically prepared answers (e.g., on MS Word, etc.) that are not typeset. If you want to submit a handwritten report, scan it and submit a PDF via Sakai. We will not accept hardcopies. If you choose to submit handwritten answers and we are not able to read them, you will not be awarded any points for the part of the solution that is unreadable.

Precision: Try to be precise. Have in mind that you are trying to convince a very skeptical reader (and computer scientists are the worst kind...) that your answers are correct.

Collusion, Plagiarism, etc.: Each team must prepare its solutions independently from other teams, i.e., without using common notes, code or worksheets with other students or trying to solve problems in collaboration with other teams. You must indicate any external sources you have used in the preparation of your solution. Do not plagiarize online sources and in general make sure you do not violate any of the academic standards of the department or the university. Failure to follow these rules may result in failure in the course.

Problem 1 (*20 points):* Trace the operation of A^* search (the tree version) applied to the problem of getting **to** Bucharest **from** Lugoj using the straight-line distance heuristic. That is, show the sequence of nodes that the algorithm will consider and the f, g, and h score for each node. You don't need to draw the graph, just right down a sequence of $(city, f(city), g(city), h(city))$ in the order in which the nodes are expanded.

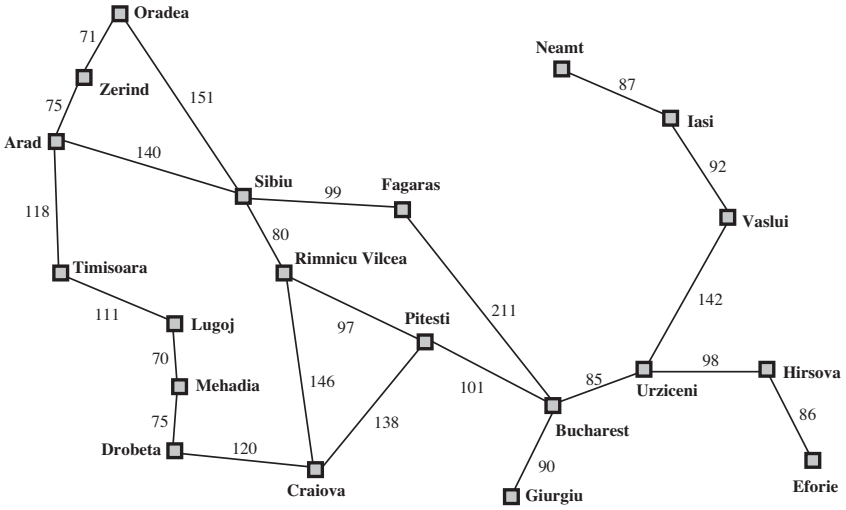


Figure 1: A simplified road map of part of Romania indicating distances between different cities.

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figure 2: Straight-line distances to Bucharest

Problem 2 (*10 points):* Consider a state space where the start state is number 1 and each state k has two successors: numbers $2k$ and $2k + 1$.

(a) Suppose the goal state is 11. List the order in which states will be visited for breadthfirst search, depth-limited search with limit 3, and iterative deepening search.

(b) How well would bidirectional search work on this problem? List the order in which states will be visited. What is the branching factor in each direction of the bidirectional search?

Problem 3 (*5 points):* Which of the following statements are correct and which ones are wrong?

(a) Breadth-first search is a special case of uniform-cost search.

(b) Depth-first search is a special case of best-first tree search.

(c) Uniform-cost search is a special case of A^* search.

(d) Depth-first graph search is guaranteed to return an optimal solution.

(e) Breadth-first graph search is guaranteed to return an optimal solution.

(f) Uniform-cost graph search is guaranteed to return an optimal solution.

(g) A^* graph search is guaranteed to return an optimal solution if the heuristic is consistent.

(h) A^* graph search is guaranteed to expand no more nodes than depth-first graph search if the heuristic is consistent.

(i) A^* graph search is guaranteed to expand no more nodes than uniform-cost graph search if the heuristic is consistent.

Problem 4 (*5 points):* Iterative deepening is sometimes used as an alternative to breadth first search. Give one advantage of iterative deepening over BFS, and give one disadvantage of iterative deepening as compared with BFS. Be concise and specific.

Problem 5 (*10 points):* Prove that if a heuristic is consistent, it must be admissible. Construct an example of an admissible heuristic that is not consistent. (Hint: you can draw a small graph of 3 nodes and write arbitrary cost and heuristic values so that the heuristic is admissible but not consistent).

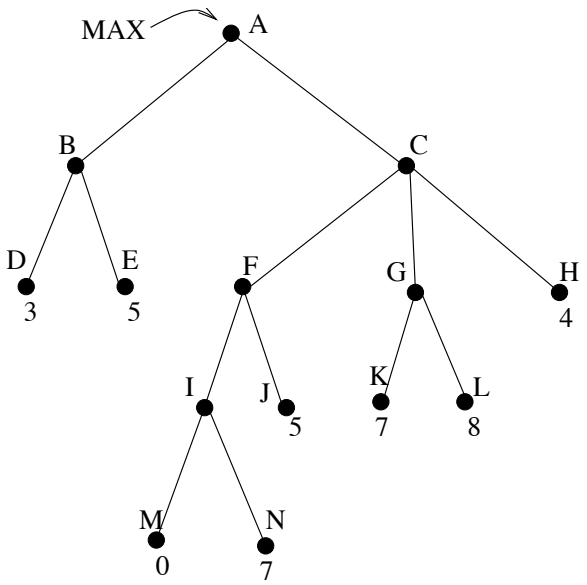
Problem 6 (*10 points):* In a Constraint Satisfaction Problem (CSP) search, explain why it is a good heuristic to choose the variable that is most constrained but the value that is least constraining.

Problem 7 (*10 points):* Consider the following game tree, where the first move is made by the MAX player and the second move is made by the MIN player.

(a) What is the best move for the MAX player using the minimax procedure?

(b) Perform a left-to-right (left branch first, then right branch) alpha-beta pruning on the tree. That is, draw only the parts of the tree that are visited and don't draw branches that are cut off (no need to show the alpha or beta values).

(c) Do the same thing as in the previous question, but with a right-to-left ordering of the actions. Discuss why different pruning occurs.



Problem 8 (*10 points):* Which of the following are admissible, given admissible heuristics h_1, h_2? Which of the following are consistent, given consistent heuristics h_1, h_2? Justify your answer.

a) $h(n) = min\{h_1(n), h_2(n)\}$

b) $h(n) = wh_1(n) + (1-w)h_2(n)$, where $0 \leq w \leq 1$

c) $h(n) = max\{h_1(n), h_2(n)\}$

Which one of these three new heuristics (a, b or c) would you prefer to use for A*? Justify your answer.

Problem 9 (*10 points)*: Simulated annealing is an extension of hill climbing, which uses randomness to avoid getting stuck in local maxima and plateaux.

a) For what types of problems will hill climbing work better than simulated annealing? In other words, when is the random part of simulated annealing not necessary?

b) For what types of problems will randomly guessing the state work just as well as simulated annealing? In other words, when is the hill-climbing part of simulated annealing not necessary?

c) Reasoning from your answers to parts (a) and (b) above, for what types of problems is simulated annealing a useful technique? In other terms, what assumptions about the shape of the value function are implicit in the design of simulated annealing?

d) As defined in your textbook, simulated annealing returns the current state when the end of the annealing schedule is reached and if the annealing schedule is slow enough. Given that we know the value (measure of goodness) of each state we visit, is there anything smarter we could do?

(e) Simulated annealing requires a very small amount of memory, just enough to store two states: the current state and the proposed next state. Suppose we had enough memory to hold two million states. Propose a modification to simulated annealing that makes productive use of the additional memory. In particular, suggest something that will likely perform better than just running simulated annealing a million times consecutively with random restarts. [Note: There are multiple correct answers here.]

(f) Gradient ascent search is prone to local optima just like hill climbing. Describe how you might adapt randomness in simulated annealing to gradient ascent search avoid trap of local maximum.

Problem 10 (*10 points)* : Consider the two-player game described in Figure 3.

1. Draw the complete game tree, using the following conventions:
2. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
3. Put each terminal state in a square box and write its game value in a circle.
4. Put *loop states* (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a "?" in a circle.
5. Now mark each node with its backed-up minimax value (also in circle). Explain how you handled the "?" values and why.

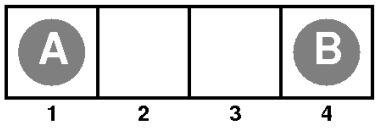


Figure 3: The start position of a simple game. Player A moves first. The two players take turns moving, and each player must move his token to an open adjacent space in *either direction*. If the opponent occupies an adjacent space, than a player may jump over the opponent to the next open space if any. (For example, if A is on 3 and B is on 2, then A may move back to 1.) The game ends when one player reaches the opposite end of the board. If player A reaches space 4 first, then the value of the game to A is +1; if player B reaches space 1 first, then the value of the game to A is -1.