1 Problem 1.

List of expanded nodes: (city, f(city), g(city), h(city))

(Lugoj,244,0,244)
(Mehadia,311,70,241)
(Lugoj,384,140,244)
(Drobeta,387,145,242)
(Craiova,425,265,160)
(Timisoara,440,111,329)
(Mehadia,451,210,241)
(Mehadia,456,215,241)
(Lugoj,466,222,244)
(Pitesti,503,403,100)
(Bucharest,504,504,0)

In the graph-version of A*, a node that has already been expanded does not get expanded again. But in the tree-version of A* (used above), a node can be expanded multiple times. The search stops whenever a node that contains the goal state has the smallest f-value in the open list.

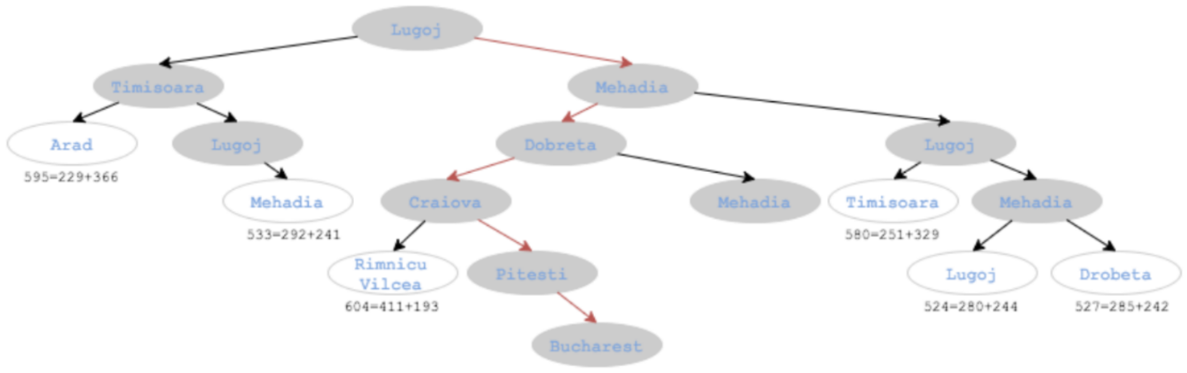


Figure 1: Diagram showing the tree version of A* (after the search has completed) applied to the problem. Expanded nodes shown in gray. List below indicates order of expansion. Subtext shows f-values of nodes in the open list.

2 Problem 2.

(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.

answer:
note: answer is also OK if you did not list repeated nodes

(a) **BFS:** 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

(b) **DLS:** 1, 2, 4, 8, 4, 9, 4, 2, 5, 10, 5, 11

(c) **IDS:** 1,1,2,1,3,1,2,4,2,5,2,1,3,6,3,7,1,2,4,8,4,9,4,2,5,10,5,11

(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?

answer:
note: slight variations on ordering, e.g. beginning with backward search OK

Bi-directional search would work very well particularly because the branching factor of the backward search is 1.
Branching factor: forward: 2, backward: 1.
State order(forward-searched nodes in bold): **1**, 11, **2**, 5, **3**, 2

3 Problem 3.

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

answer: True. BFS is UCS with cost 1 between neighboring nodes.

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

answer: True. DFS is best-first with the evaluation function f equal to the negative depth of the tree.

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

answer: True. UCS is A^* with h=0.

(d) Depth-first graph search is guaranteed to return an optimal solution.
Answer: No. Expansions in DFS depend on the order of the neighbors of each state (example: alphabetical). This could result in arbitrarily long paths.
(e) Breadth-first graph search is guaranteed to return an optimal solution.
Answer: No. If the edges have different costs, then the path returned by BFS is not guaranteed to be the shortest. It is the shortest in terms of number of states, but not when the distances between states are taken into account.
(f) Uniform-first graph search is guaranteed to return an optimal solution.
Answer: Yes.
(g) A* graph-search is guaranteed to return an optimal solution if the heuristic is consistent.
Answer: Yes.
(h) A* graph-search is guaranteed to expand no more nodes than depth-first graph search if the heuristic is consistent.
Answer: No. It is possible to conceive a simple example where DFS makes only two expansions before discovering the goal (by chance), while A* explores more branches. Note however that the path discovered by DFS may not be optimal.
(i) A* graph-search is guaranteed to expand no more nodes than uniform-first graph search if the heuristic is consistent.
Answer: No. Although Uniform-cost search is a special case of A* where the heuristic is zero everywhere, one can construct a simple example where Uniform-cost gets to the goal (by luck) sooner than an A* run that utilizes a good heuristic. On average though, A* would expand no more than uniform-cost search.

4 Problem 4.

Advantage of iterative deepening: Iterative deepening has a linear space complexity whereas BFS has an exponential space complexity. So BFS often runs out of memory quickly. Advantage of BFS: BFS is optimal in terms of number of steps on the path to the goal (not taking into account distances or edge costs), whereas iterative deepening is generally suboptimal.

5 Problem 5.

answer:

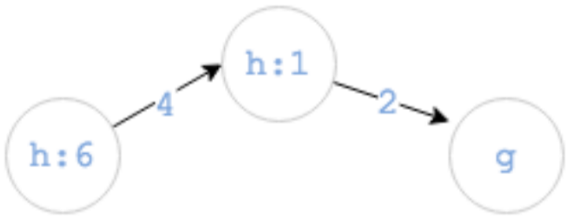


Figure 4: Values inside of nodes indicate h values, g indicates goal state, values on paths indicate costs.

We can prove this by considering a path from state s_0 to the goal state s_g, along with $c(s_i, s_{i+1})$ which is the cost between states s_i and s_{i+1}. Beginning with the consistency definition:

$$h(s_0) \leq h(s_1) + c(s_0, s_1)$$

Summing all such inequalities along the full path to the goal and canceling out common terms yields:

$$h(s_0) \leq h(s_g) + \sum_{i=1}^{g} c(s_{i-1}, s_i)$$

Where the heuristic cost of the goal state is zero:

$$h(s_0) \leq \sum_{i=1}^{g} c(s_{i-1}, s_i)$$

And the summation is equivalent to the value of $h^*(s_0)$, i.e. the actual minimum cost path to the goal:

$$h(s_0) \leq h^*(s_0)$$

6 Problem 6.

answer:
Most Constrained Variable: In the search tree, we alternate between choosing variables and choosing values for the variables. At the stage where we choose a variable, we break the search and backtrack if we find one variable that cannot be satisfied. Finding only one such variable is sufficient to say that something was wrong earlier, and we go up in the search to try other assignments. Therefore, we want to fail quickly, which will save us the trouble of trying many variables before finding the one that fails.

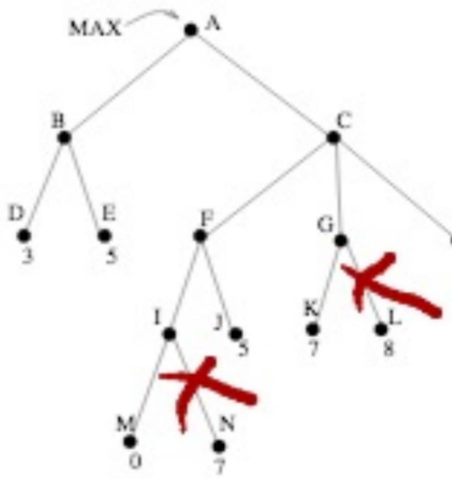
Least Constraining Value: Once we choose a variable, we have to try all the possible values before we can say that it failed. Therefore, we will be only losing time with values that fail (since we will still have to check the remaining values). But if we succeed, the search stops and we don't have the try the remaining values.

7 Problem 7.

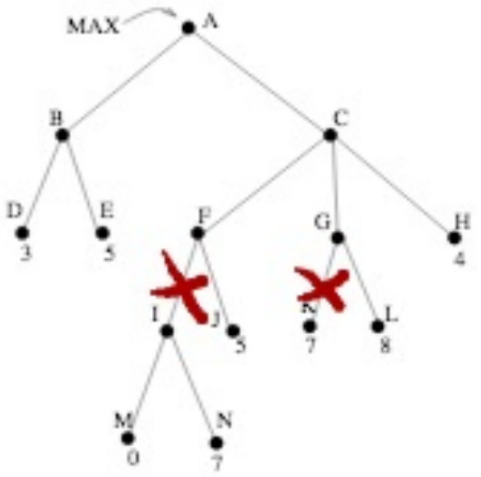
a.

The best initial move for MAX is to node C (value=4).

b.



c


MAX
A
B
C
D
3
E
5
F
G
H
4
I
J
5
K
7
L
8
M
0
N
7

8 Problem 8.

Note: In the below proofs, while proving admissibility of h, we assume admissibility of h_1 and h_2. Likewise, while proving consistency of h, we assume consistency of h_1 and h_2.

$h(n) = \min\{h_1(n), h_2(n)\}$ is admissible since given that $h_1(n) \leq h^*(n)$ and $h_2(n) \leq h^*(n)$, we can deduce that $\min\{h_1(n), h_2(n)\} \leq h^*(n)$. It is also consistent, as we will prove below:

$$\begin{aligned}
h(n) &= \min\{h_1(n), h_2(n)\} \\
&\leq \min\{h_1(n') + c(n, a, n'), h_2(n') + c(n, a, n')\} \\
&\leq \min\{h_1(n'), h_2(n')\} + c(n, a, n') \\
&\leq h(n') + c(n, a, n')
\end{aligned}$$

$h(n) = wh_1(n) + (1 - w)h_2(n)$ is admissible since given that $h_1(n) \leq h^*(n)$ and $h_2(n) \leq h^*(n)$, $wh_1(n) \leq wh^*(n)$ and $(1 - w)h_2(n) \leq (1 - w)h^*(n)$, we can add the two inequalities to obtain $h(n) = wh_1(n) + (1 - w)h_2(n)$. It is also consistent, as we will prove below:

$$\begin{aligned}
h(n) &= wh_1(n) + (1 - w)h_2(n) \\
&\leq wh_1(n') + w * c(n, a, n') + (1 - w)h_2(n') + (1 - w) * c(n, a, n') \\
&\leq wh_1(n') + (1 - w)h_2(n') + c(n, a, n') \\
&\leq h(n') + c(n, a, n')
\end{aligned}$$

$h(n) = \max\{h_1(n), h_2(n)\}$ is admissible since given that $h_1(n) \leq h^*(n)$ and $h_2(n) \leq h^*(n)$, we can deduce that $\max\{h_1(n), h_2(n)\} \leq h^*(n)$. It is also consistent, as we will prove below:

$$\begin{aligned}
h(n) &= \max\{h_1(n), h_2(n)\} \\
&\leq \max\{h_1(n') + c(n, a, n'), h_2(n') + c(n, a, n')\} \\
&\leq \max\{h_1(n'), h_2(n')\} + c(n, a, n') \\
&\leq h(n') + c(n, a, n')
\end{aligned}$$

9 Problem 9.

(a)

Hill climbing will work better than simulated annealing for problems where the fitness function has unique local extremum (that is also the global extremum). Assume a function of the form e^{-x^2} which is a parabola in 2 dimensions. It clearly has a maxima at x=0. This is the global maxima. It does not have a minima. While minimising such functions, hill climbing will work as well as simulated annealing since there would no local minima that it could get trapped into.

(b)

Randomly guessing the state will work just as well as simulated annealing for problems where the fitness function is a plateau everywhere.

(c)

Simulated annealing is a useful technique for problems where the fitness function has a lot of local extrema points. Consider the function $\frac{cos(3\pi x)}{x}$. This function has a global maximum and multiple local maxima. In such a case simulated annealing is useful to get out of local maxima that hill climbing gets into.

(d)

We should return the best (maximum or minimum) value among all the visited states, which is not necessarily the value of the state where the search stops. This is done by keeping track of the best value so far during the search.

(e)

An alternative to running simulated annealing a million times consecutively with random restarts would be to pick two million states at random and store their values into memory. Then, we generate all the neighbors of these points and keep all those that have values better than the previous pool of points. For those points that have lower values, we toss a coin and decide to add them to the new pool or to keep their parents. This process is repeated.

(f)

Combining Gradient Ascent with Simulated Annealing: Perform regular gradient ascent, but instead of always moving in the direction of the gradient, $x_{t+1} \leftarrow x_t + f'(x_t)$, sample a point $\hat{x}$ (neighbor) randomly in the vicinity of x_t so that $f(\hat{x}) < f(x_t)$ and move to it with probability $\exp\Big(t\big(f(\hat{x}) - f(x_t)\big)\Big)$. As time t increases, this probability gets smaller and eventually becomes almost zero. Which means that the algorithm initially starts by exploring the space aggressively, and as time goes by, it starts behaving in a more deterministic way by following the gradient of the objective function.