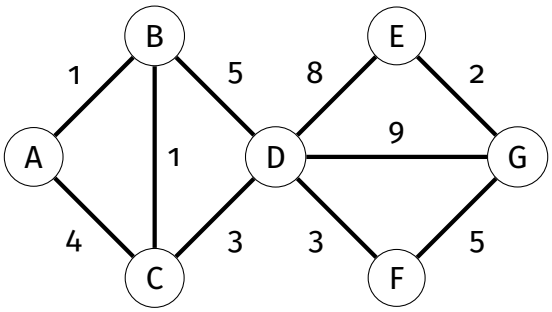
Prof. Dr. techn. Wolfgang Nejdl
Franziska Schoger

October 4th 2023

Advanced Topics in AI
Exercise 1 - Search

Question 1: Search Algorithms



Node	h_1	h_2
A	9.5	10
B	9	12
C	8	10
D	7	8
E	1.5	1
F	4	4.5
G	0	0

Consider the state space graph shown above. A is the start state and G is the goal state. The costs for each edge are shown on the graph. Each edge can be traversed in both directions. Note that the heuristic h_1 is consistent but the heuristic h_2 is not consistent.

Question 1.1: Properties

Which properties do depth-first search, breadth-first search, uniform cost search, greedy search and A* search have? Describe the approach of the methods and name the advantages and disadvantages.

Solution:

Depth-first Search: The search buffer (fringe) is organized in a last-in/first-out ("LIFO", "Stack") fashion. Solutions found are likely to be located deep in the search-tree (finds the "left-most" solution). If there are never ending paths, not all or even no solutions will be found.

Depth-first search is problematic when there are an infinite number of reachable states. In that case, the traversal of the search-tree can not be done. However, depth-first search can be implemented in a memory efficient way because it makes use of a stack data-structure.

Breadth-first Search: The fringe is organized in a first-in/first-out ("FIFO", "Queue") fashion. The shortest path inside the search-tree will be found. However,

compared to the depth-first search, the breadth-first search requires more memory ($\theta(b^d)$). This is because the number of nodes contained in the search tree is growing exponentially with respect to the depth of the tree.

Uniform Cost Search: The fringe is now ordered according to costs. It finds the least cost solution. Memory consumption is similar to Breadth-first Search.

Greedy Search: The fringe is ordered according to the heuristic, which estimates the future costs to the goal. In good cases it takes you directly to the goal, in bad cases it behaves like a badly guided DFS.

A*: The fringe is ordered according to the sum of heuristic (estimated future costs) and true costs from the start. Behaves like a guided UCS. If the heuristic is chosen well (admissible / consistent) it finds the least-cost solution.

In General: The breadth-first search is complete, since every level of the search-tree will be visited in a systematic manner and thus every node is expanded. The depth-first search on the other hand may be not complete if the search tree contains branches of infinite size. Therefore, the depth-first search may not terminate. Infinite paths through loops can be prevented by graph search. Uniform Cost Search is complete and optimal. Greedy Search is complete if there are no infinite paths, but not optimal. A* is (with admissible → tree-search, or consistent → graph-search heuristics) complete and optimal.

Question 1.2: Possible paths returned

For each of the following graph search strategies *(do not answer for tree search)*, mark which, if any, of the listed paths it could return. Note that for some search strategies the specific path returned might depend on tie-breaking behavior. In any such cases, make sure to mark all paths that could be returned under some tie-breaking scheme.

Search algorithm	A-B-D-G	A-C-D-G	A-B-C-D-F-G
Depth first search	x	x	x
Breath first search	x	x	
Uniform cost search			x
A* search with heuristic h_1			x
A* search with heuristic h_2			x

Solution:

The return paths depend on tie-breaking behaviors so any possible path has to be marked. DFS can return any path. BFS will return all the shallowest paths, i.e. A-B-D-G and A-C-D-G. A-B-C-D-F-G is the optimal path for this problem, so that UCS and A* using consistent heuristic h_1 will return that path. Although, h_2 is not consistent, it will also return this path.

Question 1.3: Heuristic function properties

Suppose you are completing the new heuristic function h_3 shown below. All the values except $h_3(B)$ are fixed and satisfy the consistency conditions.

Node	A	B	C	D	E	F	G
h_3	10	?	9	7	1.5	4.5	0

For each of the following conditions, write the set of values that are possible for $h_3(B)$. For example, to denote all non-negative numbers, write $[0, \infty)$, to denote the empty set, write $\varnothing$, and so on.

1. What values of $h_3(B)$ make h_3 admissible?

Solution:

To make h_3 admissible, h_3(B) has to be less than or equal to the actual optimal cost from B to goal G, which is the cost of path B-C-D-F-G, i.e. 12. The answer is $0 \leq h_3(B) \leq 12$

2. What values of $h_3(B)$ make h_3 consistent?

Solution:

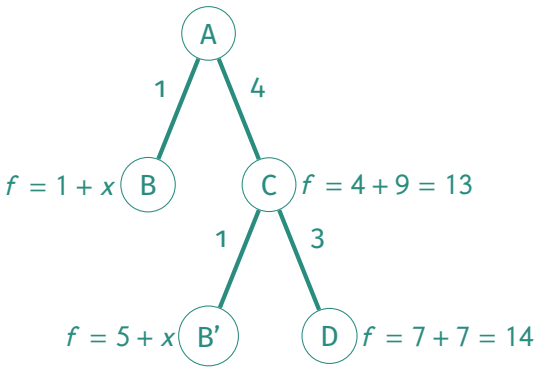
All the other nodes except node B satisfy the consistency conditions. The consistency conditions that do involve the state B are:

$h(A) \leq c(A,B) + h(B) \quad h(B) \leq c(B,A) + h(A)$
$h(C) \leq c(C,B) + h(B) \quad h(B) \leq c(B,C) + h(C)$
$h(D) \leq c(D,B) + h(B) \quad h(B) \leq c(B,D) + h(D)$

Filling in the numbers shows this results in the condition: $9 \leq h_3$ (B) ≤ 10

3. What values of $h_3(B)$ will cause A* graph search to expand node A, then node C, then node B, then node D in order.

Solution:



The A* search tree using heuristic h_3 is on the left. In order to make A* graph search expand node A, then node C, then node B, suppose h_3 (B)= x, we need

$1 + x > 13$
$5 + x < 14 \quad (expand\ B') \quad or \quad 1 + x < 14 \quad (expand\ B)$
so we can get $12 < h_3(B) < 13$

Question 2: *n*-Pac-Men search

Consider the problem of controlling n Pac-Men simultaneously. Several Pac-Men can be in the same square at the same time, and at each time step, each Pac-Man moves by at most one unit vertically or horizontally (in other words, a Pac-Man can stop, and also several Pac-Men can move simultaneously). The goal of the game is to have all the Pac-Men be at the same square in the minimum number of time steps. In this question, use the following notation: let M denote the number of squares in the maze that are not walls (i.e. the number of squares where Pac-Men can go); n the number of Pac-Men; and $p_i = (x_i, y_i) : i = 1...n$, the position of Pac-Man i. Assume that the maze is connected.

1. What is the state space of this problem?

   Solution:

   n-Tuples, where each entry is in $\{1, ..., M\}$.

2. What is the size of the state space (not a bound, the exact size)?

   Solution:

   M^n

3. Give the tightest upper bound on the branching factor of this problem.

   Solution:

   5^n (Each Pac-Man has five actions: Stop and the 4 directions).

4. Bound the number of nodes expanded by uniform cost tree search on this problem, as a function of n and M. Justify your answer.

   Solution:

   As in breadth-first search, the number of nodes expanded is bounded by b^D , with b being the branching factor, and D being the maximum depth of the search tree. Therefore, the answer is $5^{(nM)/2}$, because the max depth of a solution n is $\frac{M}{2}$ while the branching factor is 5^n . How do we know the max depth of a solution? Imagine the worst possible case: bringing together Pac-Men that are as far away from each other as possible. Since there are M total navigable cells, the maximum number of moves to accomplish this is $\frac{M}{2}$.

5. Which of the following heuristics are admissible? Which one(s), if any, are consistent? Briefly justify all your answers.

a) The number of (ordered) pairs (i, j) of Pac-Men with different coordinates:

$$h_1(p_1, ..., p_n) = \Sigma^n{}_{i=1}\Sigma^n{}_{j=i+1}\mathbf{1}[p_i \neq p_j] \quad where \quad \mathbf{1}[p_i \neq p_j] = \begin{cases} 1 & if \ p_i \neq p_j \\ 0 & otherwise \end{cases}$$

Solution:

Neither. Consider $n = 3$, no wall, and state s such that Pac-Men are at positions $(i+1, j), (i-1, j), (i, j+1)$. All Pac-Men can meet in one step, but $h(s) > 1$.

b) $h_2((x_1, y_1), ..., (x_n, y_n)) = \frac{1}{2} max \left\{ max_{i,j} \ |x_i - x_j|, max_{i,j} \ |y_i - y_j| \right\}$

Solution:

Admissible: imagine a relaxed problem where there are no walls and Pac-Men can move diagonally. The number of steps needed to solve that relaxed problem is ceil $\frac{1}{2} max(max_{i,j}|x_i - x_j|, max_{i,j}|y_i - y_j|)$. Therefore ceil h_2 is admissible. So, h_2 is also admissible, because h_2 <ceil h_2 . It is also consistent because each absolute value will change by at most 2 per step, meaning that h_2 will decrease by at most 1 for each action (actions have cost 1).

Question 3: Travelling through Romania

Perform a search to find a way through Romania. Document the process in a table using the following table schema:

Step	Fringe	Explored	Children

The columns Fringe and Explored should contain the sets at the beginning of each step, the column Children should contain the set that is created during the step. Use the street map shown in Figure 1 and the heuristic shown in Figure 2.

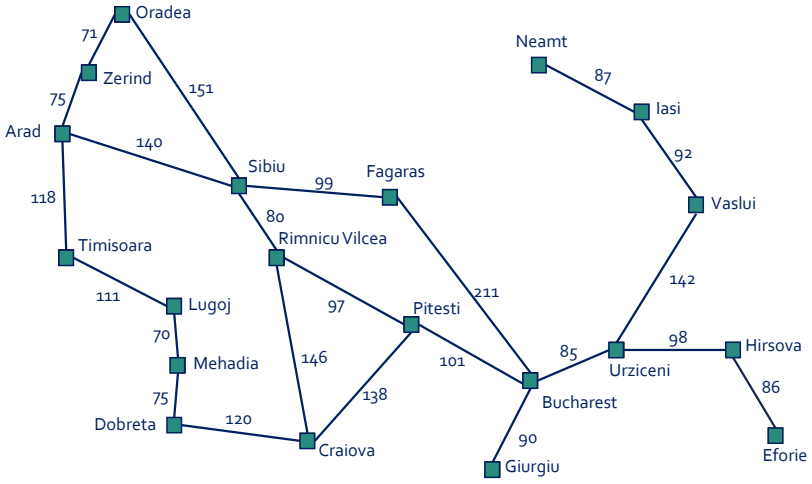


Figure 1: Simplified street map of Romania

Arad	366	Mehdia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	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: Values for the ideal distance to Bucharest

1. Perform tree-search Depth-First Search from Arad to Bucharest.

Solution:

Tree-search:

	Fringe	Explored	Children
1	Arad		Sibiu, Timisoara, Zerind
2	Sibiu, Timisoara, Zerind	Arad	Arad, Fagaras, Oradea, Rimnicu V.
3	Arad, Fagaras, Oradea, Rimnicu Vilcea, Timisoara, Zerind	Arad, Sibiu	Sibiu, Timisoara, Zerind
4	Sibiu, Timisoara, Zerind, Fagaras, Oradea, Rimnicu Vilcea, Timisoara, Zerind	Arad	Arad, Fagaras, Oradea, Rimnicu V.

and so on and so forth. With tree-search we are stuck in a loop. So let's do graph-search.

	Fringe	Explored	Children
1	Arad		Sibiu, Timisoara, Zerind
2	Sibiu, Timisoara, Zerind	Arad	Arad, Fagaras, Oradea, Rimnicu V.
3	Arad, Fagaras, Oradea, Rimnicu Vilcea, Timisoara, Zerind	Arad, Sibiu	
4	Fagaras, Oradea, Rimnicu Vilcea, Timisoara, Zerind	Arad, Sibiu	**Bucharest**, Sibiu
5	**Bucharest**, Sibiu, Oradea, Rimnicu Vilcea, Timisoara, Zerind	Arad, Sibiu, Fagaras	

What happens now in step 3 is that we notice that we explored Arad already and not explore it again. So in step 4 we explore the next city in the fringe, Fagaras.

2. Perform tree-search Breadth-First Search from Arad to Bucharest.

Solution:

	Fringe	Explored	Children
1	Arad		Sibiu, Timisoara, Zerind
2	Sibiu, Timisoara, Zerind	Arad	Arad, Fagaras, Oradea, Rimnicu V.
3	Timisoara, Zerind,Arad, Fagaras, Oradea, Rimnicu Vilcea	Arad, Sibiu	Arad, Lugoj
4	Zerind,Arad, Fagaras, Oradea, Rimnicu Vilcea, Arad, Lugoj	Arad, Sibiu, Timisoara	Arad, Oradea
5	Arad, Fagaras, Oradea, Rimnicu Vilcea, Arad, Lugoj, Arad, Oradea	Arad, Sibiu, Timisoara, Zerind	Sibiu, Timisoara, Zerind
6	Fagaras, Oradea, Rimnicu Vilcea, Arad, Lugoj, Arad, Oradea, Sibiu, Timisoara, Zerind	Arad, Sibiu, Timisoara, Zerind,Arad	**Bucharest**, Sibiu
7	Oradea, Rimnicu Vilcea, Arad, Lugoj, Arad, Oradea, Sibiu, Timisoara, Zerind, **Bucharest**	Arad, Sibiu, Timisoara, Zerind,Arad, Fagaras	Sibiu, Zerind
8	Rimnicu Vilcea, Arad, Lugoj, Arad, Oradea, Sibiu, Timisoara, Zerind, **Bucharest**, Sibiu, Zerind	Arad, Sibiu, Timisoara, Zerind, Arad, Fagaras, Oradea	Craiova, Pitesti, Sibiu
9	Arad, Lugoj, Arad, Oradea, Sibiu, Timisoara, Zerind, **Bucharest**, Sibiu, Zerind, Craiova, Pitesti, Sibiu	Arad, Sibiu, Timisoara, Zerind, Arad, Fagaras, Oradea, Rimnicu Vilcea	Sibiu, Timisoara, Zerind
10	Lugoj, Arad, Oradea, Sibiu, Timisoara, Zerind, **Bucharest**, Sibiu, Zerind, Craiova, Pitesti, Sibiu, Sibiu, Timisoara, Zerind	Arad, Sibiu, Timisoara, Zerind, Arad, Fagaras, Oradea, Rimnicu Vilcea, Arad	Meha-dia, Timisoara
11	Arad, Oradea, Sibiu, Timisoara, Zerind, **Bucharest**, Sibiu, Zerind, Craiova, Pitesti, Sibiu, Sibiu, Timisoara, Zerind, Mehadia, Timisoara	Arad, Sibiu, Timisoara, Zerind, Arad, Fagaras, Oradea, Rimnicu Vilcea, Arad, Lugoj	Sibiu, Timisoara, Zerind

12	Oradea, Sibiu, Timisoara, Zerind, **Bucharest**, Sibiu, Zerind, Craiova, Pitesti, Sibiu, Sibiu, Timisoara, Zerind, Mehadia, Timisoara, Sibiu, Timisoara, Zerind	Arad, Sibiu, Timisoara, Zerind, Arad, Fagaras, Oradea, Rimnicu Vilcea, Arad, Lugoj, Arad	Sibiu, Zerind
13	Sibiu, Timisoara, Zerind, **Bucharest**, Sibiu, Zerind, Craiova, Pitesti, Sibiu, Sibiu, Timisoara, Zerind, Mehadia, Timisoara, Sibiu, Timisoara, Zerind, Sibiu, Zerind	Arad, Sibiu, Timisoara, Zerind, Arad, Fagaras, Oradea, Rimnicu Vilcea, Arad, Lugoj, Arad, Oradea	Fagaras, Oradea, Rimnicu V.
14	Timisoara, Zerind, **Bucharest**, Sibiu, Zerind, Craiova, Pitesti, Sibiu, Sibiu, Timisoara, Zerind,Mehadia, Timisoara, Sibiu, Timisoara, Zerind, Sibiu, Zerind, Fagaras, Oradea, Rimnicu V.	Arad, Sibiu, Timisoara, Zerind, Arad, Fagaras, Oradea, Rimnicu Vilcea, Arad, Lugoj, Arad, Oradea, Sibiu	Arad, Lugoj
15	Zerind, **Bucharest**, Sibiu, Zerind, Craiova, Pitesti, Sibiu, Sibiu, Timisoara, Zerind, Mehadia, Timisoara, Sibiu, Timisoara, Zerind, Sibiu, Zerind, Fagaras, Oradea, Rimnicu V., Arad, Lugoj	Arad, Sibiu, Timisoara, Zerind, Arad, Fagaras, Oradea, Rimnicu Vilcea, Arad, Lugoj, Arad, Oradea, Sibiu,Timisoara	Arad, Oradea
16	**Bucharest**, Sibiu, Zerind, Craiova, Pitesti, Sibiu, Sibiu, Timisoara, Zerind, Mehadia, Timisoara, Sibiu, Timisoara, Zerind, Sibiu, Zerind, Fagaras, Oradea, Rimnicu V., Arad, Lugoj	Arad, Sibiu, Timisoara, Zerind, Arad, Fagaras, Oradea, Rimnicu Vilcea, Arad, Lugoj, Arad, Oradea, Sibiu,Timisoara, Zerind	

Two comments:

a) We do a lot of work multiple times. Look at all those gray entries in the table. We already explored them before, but we explore them again and again. In Graph Search we could just ignore these and would have been done in 9 steps instead of 16.

b) With BFS you can do an early goal test and check the child nodes and stop as soon as a child is a goal state (here step 6). Since that state will be the first goal en-queued, it will also be the first de-queued (FIFO queue). So you can save on expanded nodes. We didn't do that in the lecture, where we only did a goal test, when we de-queued a node. So we have to wait until Bucharest is on top of the Fringe (step 16). For all other algorithms this early goal test does not work.

3. Perform tree-search Uniform Cost Search from Sibiu to Bucharest.

Solution:

	Fringe	Explored	Children
1	Sibiu		Arad(140), Oradea(151), Fargaras(99), Rimnicu Vilcea(80)
2	Rimnicu Vilcea(80), Fargaras(99), Arad(140), Oradea(151)	Sibiu	Pitesti(177), Craiova(226), Sibiu(160)
3	Fargaras(99), Arad(140), Oradea(151), Pitesti(177), Craiova(226)	Sibiu, Rimnicu Vilcea	**Bucharest(310)**, textcolorgraySi-biu(198)
4	Arad(140), Oradea(151), Pitesti(177), Craiova(226), **Bucharest(310)**	Sibiu, Rimnicu Vilcea, Faragas	Zerind(215), Timisoara(258), Sibiu(280)
5	Oradea(151), Pitesti(177), Zerind(215), Craiova(226), Timisoara(258), **Bucharest(310)**	Sibiu, Rimnicu Vilcea, Faragas, Arad	Zerind(222), Sibiu(302)
6	Pitesti(177), Zerind(215), Craiova(226), Timisoara(258), **Bucharest(310)**	Sibiu, Rimnicu Vilcea, Faragas, Arad, Oradea	Bucharest(278), Craiova(315), Rimnicu Vilcea(274)
7	Zerind(215), Craiova(226), Timisoara(258), **Bucharest(278)**	Sibiu, Rimnicu Vilcea, Faragas, Arad, Oradea, Pitesti	Oradea(286), Arad(290)
8	Craiova(226), Timisoara(258), **Bucharest(278)**	Sibiu, Rimnicu Vilcea, Faragas, Arad, Oradea, Pitesti, Zerind	Dobreta(346), Rimnicu Vilcea(372), Pitesti(364)
9	Timisoara(258), **Bucharest(278)**, Dobreta(346)	Sibiu, Rimnicu Vilcea, Faragas, Arad, Oradea, Pitesti, Zerind, Craiova	Lugoj(369), Arad(376)
10	**Bucharest(278)**, Dobreta(346), Lugoj(369)	Sibiu, Rimnicu Vilcea, Faragas, Arad, Oradea, Pitesti, Zerind, Timisoara	

As we have seen with BFS how large this table can become if we are doing tree-search, graph-search is given here. If you want to do tree-search, add the children given in gray to the fringe at the correct place.
One interesting thing we can see happening from step 5 to step 6 and from step 6 to step 7. In step 5 we have Zerind as Child again. We haven't been

to Zerind yet, so neither with tree-search nor with graph-search we can ignore it. Straightforward we could add Zerind again to the fringe, at the respective place (between the Zerind already on the fringe and Craiova). If we then encounter it, we would check with graph search if Zerind was already explored and then go on to the next node. With tree search we would expand it again. Here we chose another method. We updated the fringe. If a node is already on the fringe, we update it by keeping only the version with the lower costs. For Zerind, the version already on the fringe has the lowest costs, so it stays unchanged. From step 6 to step 7, the newly found path to Bucharest has the lower costs, so we update the costs on the fringe (and also the position if necessary). The result is the same as adding the new node at the correct position.

4. Perform tree-search A* Search from Arad to Bucharest

Solution:

Step	Fringe	Explored	Children
1	Arad		Sibiu (140 + 253 = 393), Timisoara (118 + 329 = 447), Zerind (75 + 374 = 449)
2	Sibiu (393), Timisoara (447), Zerind (449)	Arad	Rimnicu Vilcea (140 + 80 + 193 = 413), Fagaras (140 + 99 + 176 = 415), Arad (140 + 140 + 366 = 646), Oradea (291 + 380 = 671)
3	Rimnicu Vilcea (413), Fagaras (415), Timisoara (447), Zerind (449), Oradea (671)	Arad, Sibiu	Pitesti (220 + 97 + 100 = 417), Craiova (220 + 146 + 160 = 526), Sibiu (220 + 80 + 253 = 553)
4	Fagaras (415), Pitesti (417), Timisoara (447), Zerind (449), Craiova (526), Oradea (671)	Arad, Sibiu, Rimnicu Vilcea	**Bucharest (239 + 211 + 0 = 450)**, Sibiu (239 + 99 + 253 = 591)
5	Pitesti (417), Timisoara (447), Zerind (449), **Bucharest (450)**, Craiova (526), Oradea (671)	Arad, Sibiu, Rimnicu Vilcea, Fagaras	Craiova (317 + 138 + 160 = 615), **Bucharest (317 + 101 + 0 = 418)**, Rimnicu Vilcea (317 + 97 + 193 = 607)
6	**Bucharest(418)**, Timisoara (447), Zerind (449), Craiova (526), Oradea (671)	Arad, Sibiu, Rimnicu Vilcea, Fagaras, Pitesti	

Here we did again graph search. If you want to do tree search, add the gray children at the correct position of the fringe.
From step 5 to step 6, we updated again the value of Bucharest, which moves it to the top of the fringe.