EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATH. - NATURWISS. FAKULTÄT
FACHBEREICH INFORMATIK
KOGNITIVE SYSTEME · PROF. A. ZELL

Artificial Intelligence

Assignment 3

Assignment due on: 19.11.2021

Question 1 Heuristics (2 + 1 = 3)

Consider a shortest path problem where the costs between two nodes are defined by the travel distance.

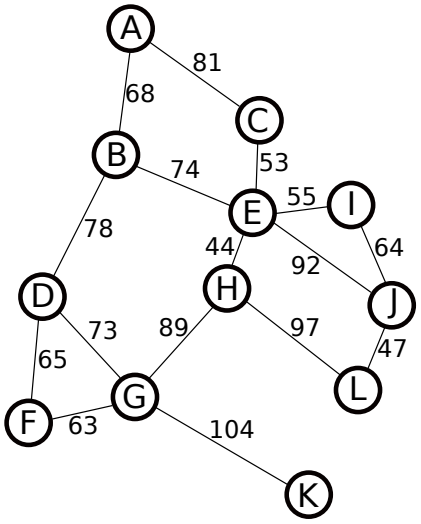
(a) For each of the following heuristics, determine whether it is guaranteed to be admissible and/or consistent:

$h(n) = ...$

- straight-line-distance
- $c *$ straight-line-distance , $c \in \mathbb{R}^+$ (c: constant over all n)
- Manhattan distance (i.e. latitude distance + longitude distance)
- latitude distance

(b) Sketch a small example with at most four nodes which displays an admissible but not consistent heuristic.

Question 2 Greedy best-first search (3 + 1 = 4)

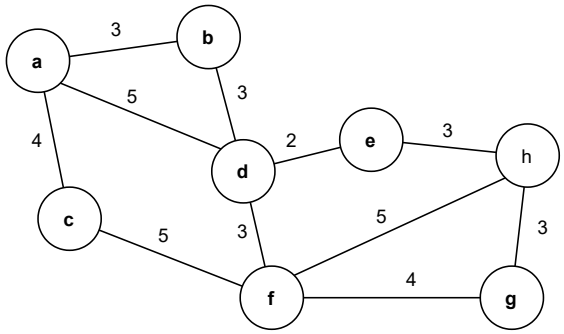


n	$h(n)$
A	216
B	169
C	167
D	145
E	125
F	128
G	82
H	100
I	139
J	90
K	0
L	54

(a) Use Greedy best-first search with the heuristic given in the table to find a path between the node $\mathbf{A}$ and the target $\mathbf{K}$. Write down all expanded nodes in correct order and the final resulting path. Is the resulting path optimal? Justify your answer.

(b) In fact, the given graph models some significant cities in Baden-Württemberg with the travel costs between them. The given heuristic is the straight-line-distance between each city and the goal city. Consider to use the latitude distance as heuristic instead. In general, would you expect a higher or smaller effective branching factor? Why?

Question 3 Pathfinding with A* (5 points)



n	$h(n)$
a	11
b	5
c	8
d	4
e	2
f	3
g	2
h	0

Note: For the nodes with equal path cost i.e. f-values, expand them in the alphabetic order. For part (a) of this question fill in the table like following for building your solutions:

node	g	h	f	priority queue	(expansion node, f-value, predecessor)
a	0	11	11	(a, 11)	(a, 11, -)
b	3	5	8	(b, 8)	
c	4	8	12	(b, 8) (c, 12)	
d	5	4	9	(b, 8) (d, 9) (c, 12)	(b, 8, a)
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

(a) Find the shortest path between the node **a** and the target node **h** using A* by hand for the graph above. Detail each intermediate step and indicate the f-values. (3 point)

(b) Is there a unique shortest path for the above graph? If not indicate all other path(s). (1 point)

(c) What happens, if we change the heuristic of node f to be h(f) = 6. (1 point)

Question 4 Uniformed vs. Informed Search (8 points)

In this exercise we will implement both Dijkstra's algorithm (uniform cost search) and A* graph search in Python. When running any of these we need to keep track of two sets: the *frontier*, a priority queue of nodes that we need to visit, and *explored*, the set of nodes that we have already visited and thus don't want to visit again. These sets can be easily implemented as a *list* in Python.

The sample file *assignment03.py* contains a class for defining German cities. The properties of the class holds the information about the distances (when traveling by car) between the neighboring cities, the coordinates of each city, and some other useful variables. It also contains a method `g(n)` which gives the distance between the current city and a particular neighbor. The function `initialize_all_cities()` defines a dictionary containing a graph of all German cities. Each item in the dictionary holds an instance of the class, containing information about a node in the graph.

(a) Implement a reasonable admissible heuristic function `h(n)` for A* graph search based on the geographical data available for the graph as a method in `DefineCity`. (`h(n) = 0` is not an acceptable answer for this question). *(1 point)*

(b) Implement a function `find_path((start_name, goal_name, search_mode))`, which searches the lowest-cost path from start to goal. `start_name` and `goal_name` are the names of cities defined in the python file. `search_mode` can be either `'a*'` or `'dijkstra'`. This function should then implement the corresponding algorithm. Both algorithms accumulate the path cost g. However, A* then uses $f = g + h$ for its decisions. To implement it in a simple way, make sure f equals the accumulated path cost for Dijkstra's algorithm.
It returns three objects: the computed *path* (as a list), the *cost* for this path, and the set of *explored cities* (also as a list), while computing this path.
You can use slide 25 from chapter 3 as a template for implementing this. Briefly summarized: *(5 points)*

- It should expand a city and calculate the cost (depending on `search_mode`) for all its successors. The expanded city must be removed from the *frontier* and added to the *explored* list. All the successors (not in *explored*) must be added to the *frontier* list. Make sure that the costs `f`, `sum_g` and the parent node are updated inside the object.
- It should choose the next city to be expanded from the *frontier* list having the minimum cost `f(n)`.
- The above two steps should be repeated until it reaches the goal city.
- It should return the path from the start to the goal city by using the parents nodes saved in the city objects.
- Check the algorithm for the pair of cities provided in the template.

Hint: Save copies of the city object in the frontier using `copy.deepcopy()` rather than the original object. This makes sure that the cost of cities are not overwritten accidentally.

(c) What is the difference between A* and Dijkstra? *(1 point)*

(d) Compare the set of explored cities for the example tasks from the template. What do you observe? Explain your observation briefly! *(1 point)*