EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

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

Artificial Intelligence

Assignment 4

Assignment due on: 26.11.2021

Question 1 Hill Climbing Search (1+1+2+1+2+1 = 8 points)

In this exercise we will have a look at the n-queens problem (moving only one queen at a time along its column) and Hill Climbing Search. The representation in python is a list of length n with integers between 0 and $n-1$, symbolizing the row-position of each queen. Fill in the #TODOs in $assignment04\_1.py$.

(a) Implement a heuristic function `h(state)` that returns the number of queen pairs which are on the same row or diagonal (columns are impossible by our above problem statement).

(b) Implement the function `generate_successors(state)` which returns all successor states for a given state, not just the ones with less `h`-value.

(c) Implement the function `hill_climb(start)`. It implements the main search loop, calling `generate_successors` and choosing by their respective `hvalues`. It returns `path_to_goal`, the list of states which are visited during the search. The first one is the start state, the last one the goal state (if reached). Also, it returns `reached_goal`, which evaluates to `True` iff the final state is a goal state.

(d) Find a local optimum of `h` for the 4-queens problem which is not a goal state.

(e) Now modify this function to `hill_climb(start, double_step=False)`. If `double_step=False`, it will execute as before. If `double_step=True`, it always does two steps at once, i.e. given a state it considers a successor as one where two queen moves were made. Now apply this function to the local optimum you found in (d).

(f) Find a local optimum for the 8-queens problem with `double_step=True` which is not a goal state.

Question 2 Recursive Best-First Search (RBFS) (2+4 = 6 points)

For this question draw the trees for recursive best-first search. Please represent each node as a circle with its name inside, write the f_limit in a rectangle above the corresponding node, and express the $f-value$ under each node. The solution path should be highlighted. (hint: follow the algorithm shown in slide 56, chapter 3)

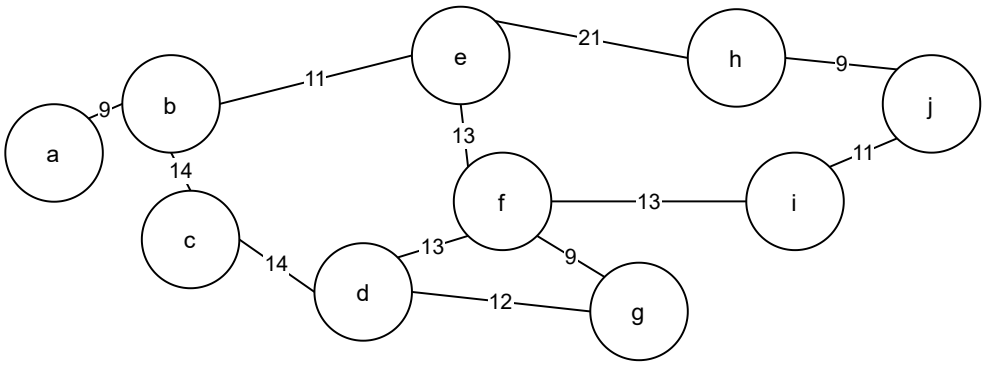


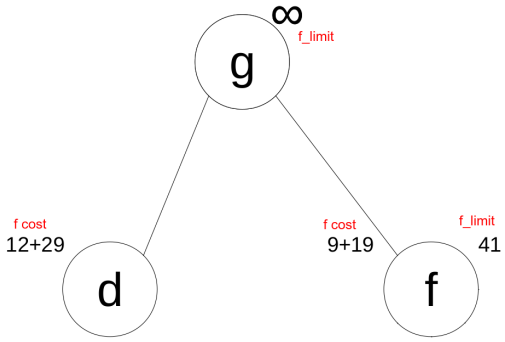
Table 1.

n	$h(n)$	n	$h(n)$
a	37	f	19
b	31	g	26
c	36	h	7
d	30	i	9
e	22	j	0

Table 2.

n	$h(n)$	n	$h(n)$
a	33	f	19
b	26	g	26
c	33	h	0
d	29	i	12
e	17	j	7

(a) Using the heuristic function shown in Table 1, find a path from node **a** to the target node **j**. *Note: See the following graph as hint, to build the complete solution.*



(b) Using the heuristic function shown in Table 2, find a path from node **g** to the target node **h**.

Question 3 RBFS Programming in Python (5+1 = 6 points)

We want to implement the recursive best-first search (RBFS) algorithm in this task. Use the template *assignment04_3.py* for this task. Similar to last week, it contains a graph of German cities defined using the `DefineCity` class.

Note: In the template, the places where you must implement the code are commented with `#TODO` tags. Comments and details for implementation are also included in the template.

(a) Implement a function `rbfs(start, goal, f_limit, count=0)` which follows the complete algorithm as shown in Figure 3.26 in the book. It should take in the `start_city`, `goal_city`, `f_limit`, and optionally `count`. It will return whether the search was successful, the final city object, and total number of visited cities. In detail:

- Perform the goal test at the start.

- Create the list of successors from `start.neighbours` and set all cost values accordingly. *Hint: Use* `copy.copy()` *to not override entries from* `cities`.
- Implement the main loop including the decision for the next city to be expanded, the check `min_f > f_limit`, the recursive call with new `f_limit`, and the `return` if the recursive call was successful.

(b) Implement a function `graphsearch_rbfs(start_name, goal_name)` which calls above function with the correct cities, then unrolls the path to the goal city and returns it as a list.