CSPB 3202: Introduction to Artificial Intelligence
Homework 2A

Learning objective: Use heuristics to solve search problems; use back-forward iteration to solve the connected-graph color constraint problem.

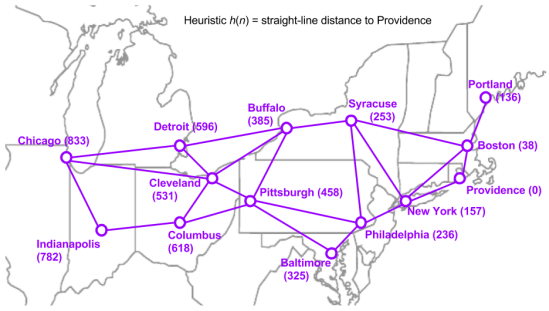
1 Neal Page is done with Chicago

Neal Page has had it up to **HERE** with living in Chicago, so far away from where he has to do most of his business. He's decided to pack up his family and move to Providence, this time in the middle of summer. Because we all know that summertime in the northwest corridor is absolutely peaches.

Del is a close friend of the family now and has decided to help his good friend Neal make the move. Who knows, maybe he'll live in Providence too? Neal is really only in it for the map of straight-line distances to Providence that Del has meticulously created, as shown graphically in **sld_providence** below. Even more valuable is the **sld_providence** dictionary in the skeleton code, which Del wrote because he is also a script kiddie. Note that because it doesn't make much sense to add estimated travel times and the straight-line distance, we will only be using the **map_distances** state space graph for this problem.

map_distances

sld_providence



1.1 A-star Search

Modify your code for uniform-cost search from Homework1A so that it provides optionally as output the number of nodes *expanded* in completing the search.

Include a new optional logical (True/False) argument **return_nexp**, so your function calls to the new uniform cost search will look like: **uniform_cost(start, goal, state_graph, return_cost, return_nexp)**.

- If **return_nexp** is True, then the last output in the output tuple should be the number of nodes expanded.
- If **return_nexp** is False, then the code should behave exactly as it did in Homework1A.

Then, verify that your revised codes are working by checking Neal's optimal route from New York to Chicago. Include the number of nodes expanded and the path cost (using **map_distances**).

Moodle Quiz Problem 1. Pass the UCS unit tests (see skeleton code).

1.2 Heuristic Function

Define a function to take as an argument the state that Neal is in (city on our graphs), and return as output the value of the straight-line distance heuristic, between Neal's state and Providence. Note that your function should be quite short, and amounts to looking up the proper value from the **sld_providence** dictionary defined in the helper functions. Call this function **heuristic_sld_providence**.

This might seem sort of silly right now, but it will make your codes much easier to adapt in Problem 3.

Moodle Quiz Problem 2. Pass the heuristic function lookup unit tests (see skeleton code).

1.3 Full-blown A*

We are finally ready to help Neal use his knowledge of straight-line distances from various cities to Providence to inform his family's route to move from Chicago to Providence!

Modify your uniform-cost search codes from **1.1** even further so that they now perform **A* search**, using as the heuristic function the straight-line distance to Providence.

Provide **heuristic** as an additional argument, which should just be the function to call within the A* code. So your call to the A* routine should look like: **astar_search(start, goal, state_graph, heuristic, return_cost, return_nexp)**. (This kind of modular programming will make it much easier to swap in alternative heuristic functions later, and also helps to facilitate debugging if something goes wrong.)

Moodle Quiz Problem 3. Pass the **astar_search** unit tests (see skeleton code).

1.4 Middle-of-the-Journey test

Neal and Del have already made it to Buffalo, but there is a freak snow storm up there (as there always is!) and they're disoriented from the lack of luck. Use your A* search code from **1.3** to help them find the optimal path by distance traveled the rest of the way to Providence.

Moodle Quiz Problem 4. Print the the following using your code:

1. the optimal path
2. the optimal path cost (miles traveled)
3. the number of states expanded during the A* search

Additionally, print how many states must be expanded to find the optimal path from Buffalo to Providence using the regular old uniform-cost search algorithm from **1.1**.

Moodle Quiz Problem 5. Comment on the difference in states that must be explored by each algorithm. Sanity check: No matter what your start and goal states are, how should the output from **astar_search** and **uniform_cost** search compare?

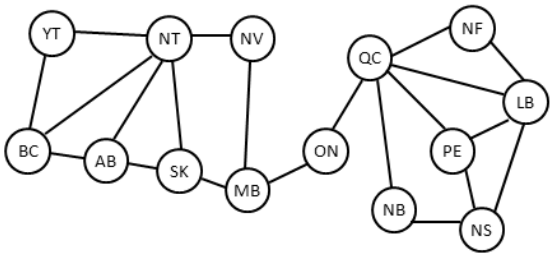
Moodle Quiz Problem 6. How many states are expanded by each of A* search and uniform cost search to find the optimal path *from Philadelphia to Providence*?

2 O, Canada!

We've discussed how map coloring problems are prime examples of constraint satisfaction problems. In this exercise you'll have the chance to test out backtracking, constraint propagation, and leveraging structure. See below for a map we're interested in coloring of mainland Canada.

colored_map

graphical_map



In this problem, you only have the colors ("values") **red, green**, and **blue** (why do we need 3 colors?). In order to solve the constraint satisfaction problem over **graphical_map**, write a class **CSP(vars, neighbors, domain)** where **vars** is a list of variables which must have values from the list **domain** assigned to them, **neighbors** is a dict of **var:[var,...]** that for each variable lists the other variables that cannot have the same value. This class will be passed to different solver algorithms that you will write. Note that in this constraint satisfaction problem we implicitly consider the constraints to be that adjacent variables cannot hold the same value.

Moodle Quiz Problem 7. Pass the unit tests for the CSP class (see skeleton code).