Sleeps with the Fishes

CMS 430, Spring 2017

Due Tuesday April 4 at 11:59 PM

You may work with a partner on this project.

Subset Sum

Given a list of integers, is there a subset that sums to a given target value? For example, for the target 29 and the list

$$[2, 8, 4, 1, 5, 11, 7, 13]$$

the answer is *yes*, because $11+13+5$ sums to 29. There may be multiple subsets that satisfy the problem: $2 + 8 + 1 + 5 + 13$ also sums to 29.

Use the included file `subset_sum.py` to write a program that uses the DEAP genetic algorithm library to solve instances of the subset sum problem. The code randomly generates a list of 100 integers in than range 1 to 100 and a random target between 1 and 1000. Each individual in the population is a vector of 100 binary values; a 1 indicates that the corresponding number is included in the sum and 0 indicates that it is not.

Your job is to write a fitness function that will allow the GA to identify the members of each list that sum up to the target value.

This problem is not very hard, but here are a few tips:

- If you don't have DEAP installed from our in-class lab, use `sudo pip install deap` to get it.
- You will need to decide if your fitness function corresponds to minimizing or maximizing an objective value. Look at the top three lines and choose either `FitnessMax` or `FitnessMin` on the third line.
- you can experiment with tweaking the algorithm's parameters to achieve convergence on every trial if you need to.

Who Owns the Fish?

This logic puzzle is often attributed to Einstein, who (allegedly) claimed that fewer than 2% of people would be able to solve it. But now, using the power of search, you'll be able to join that (possibly nonexistent) elite club.

The Puzzle

There are five houses in a row, each painted a different color. In each house lives a person with a different nationality. The five owners each drink a different drink, smoke a different brand of tobacco, and keep a different pet. One of the pets is a walleye pike.

1. The Brit lives in the red house.
2. The Swede keeps dogs as pets.
3. The Dane drinks tea.
4. The green house is on the left of and next to the white house.
5. The green house owner drinks coffee.
6. The person who smokes Pall Malls keeps birds.
7. The owner of the yellow house smokes Dunhills.
8. The man living in the house right in the center drinks milk.
9. The man who smokes Blends lives next to the one who keeps cats.
10. The Norwegian lives in the first house.
11. The man who keeps horses lives next to the one who smokes Dunhills.
12. The owner who smokes Bluemasters drinks beer.
13. The German smokes Princes.
14. The Norwegian lives next to the blue house.
15. The man who smokes Blends has a neighbor who drinks water.

Who owns the fish?

Solving It

First, get a mental model of the five houses sitting in a line. Got it? Okay.

Think of each house as a basket of five attributes: color, nationality, drinks, smokes, and pet. Each property has five different possible values

- *color* can be red, blue, white, yellow, or green
- *nationality* can be Brit, German, Swede, Dane, or Norwegian
- *drinks* can be beer, water, milk, tea, or coffee
- *smokes* can be Dunhills, Princes, Bluemasters, Blends, or Pall Malls
- *pet* can be dogs, horses, birds, cats, or **THE FISH**

Therefore, all you need to do is search for a combination of assignments to the 25 variables that satisfies the puzzle's 15 constraints.

This kind of problem, where there is no objective function to minimize, but a set of restrictions to satisfy, is called a *constraint satisfaction problem*.

I recommend using a recursive depth-first search for this problem, because it avoids the need to keep a large number of search nodes in memory. Here is a pseudocode implementation of a search routine:

```
def search(houses):
  house, attribute = find a house and one of its attributes that
                     is set to None

  if there are no remaining attributes set to None:
    print the solution and exit

  for each possible value of attribute:
      if assigning the value is valid:
          houses[house][attribute] = value
          search(houses)
          houses[house][attribute] = None  # Clear the assignment

  return
```

In this setup, most of the work is in checking that a proposed assignment is valid—that is, that it does not violate any of the fifteen constraints laid out for the puzzle.

Setup

You can represent the houses as a list of dictionaries:

```
houses = [{} for i in range(5)]
for house in houses:
    house['color'] = None
    house['nationality'] = None
    house['drinks'] = None
    house['smokes'] = None
    house['pet'] = None
```

Some facts are readily apparent from the constraints:

```
# The man living in the house right in the center drinks milk.
houses[2]['drinks'] = 'milk'

# The Norwegian lives in the first house
houses[0]['nationality'] = 'Norwegian'

# The Norwegian lives next to the blue house
houses[1]['color'] = 'blue'
```

Each of the constraints can be implemented as a function that tests if the constraint is violated or not. For example, constraint 1 could look like this:

```
def brit_lives_in_red_house(houses):
  for h in houses:

      # At least one value is unassigned---skip this house
      if h['nationality'] is None or h['color'] is None:
          continue

      # One value is either Brit or red, but the other is not
      if h['nationality'] == 'Brit' and h['color'] != 'red':
          return False

      if h['nationality'] != 'Brit' and h['color'] == 'red':
          return False

  # No house contradicts the test
  return True
```

Most of your work will be in writing these constraints, but most of them have a nearly identical form.

Strategy

I recommend starting simple by first implementing the basic depth-first search with no constraints. This will fill in properties for every house and then exit—the solution will be wrong, but you'll be sure that the search can work.

Now, add in functions for the constraints one at a time, checking each time that the solution adapts to incorporate the new constraint. When you have all constraints in place, you'll have the answer.