Project 2: Local Search Optimization

Max has recently started his own manufacturing company that has grown to employ several people. He has several duties as the owner, including that he creates the schedules for X employees over the next Y days. Each day has three shifts: morning, evening, and graveyard. X will always be less than Y. For simplicity's sake, you can also assume that Y will be a multiple of X.

When there are few employees to schedule over a small number of days, Max has a fairly easy time writing up this schedule. He wants to be able to produce much longer schedules, but he has been having trouble satisfying the conditions of his employees and the government regulations he must follow. He has pulled several all-nighters trying to produce schedules to satisfy the constraints, but he has had no luck.

Since you have expert knowledge in local search optimization, Max has outsourced this work to you. Your task is to produce valid schedules using local search. You decide to implement two different local search algorithms, and a few different heuristics to boot.

Step 0: Read the starter code

You will be given two files that describe a schedule and the days within. Read these files to understand how they work. The string representation of a schedules uses the python module prettytable, which you can install on your local machine with this command (not required to complete the project):

```
sudo pip3 install --user prettytable
```

Step 1: Implement the heuristics

To get started with using the provided code, you will implement a few heuristics. They are given to you as stubs first. There are two sample schedules in the starter code, along with a file called "values.txt." You can load these schedules to test your value functions.

value1:

A schedule is considered bad if it contains a day where the same employee works more than once. This heuristic should count the number of same-day pairs of shifts that are not the same, and return that number. Return the value as an integer.

value2:

A schedule is considered bad if there are (1) employees who work on the same day, (2) even numbered employees working on the same day as odd numbered employees. Reuse what you did in value1, but this time count the number of days without an even/odd employeees mix as well. Return the value you calculate as an integer.

value3:

A schedule is considered bad if there are (1) employees who work on the same day, (2) even numbered employees working on the same day as odd numbered employees, (3) employees who work a graveyard shift followed by a morning shift, (4) a schedule isn't balanced, meaning that employes are scheduled evenly over the number of days if you have five employees over 10 days every employee would be scheduled six times for that schedule to be balanced. Count the number numer of employees who are evenly scheduled. Also, count the number of times in the schedule where the morning shift immediately folllowing a graveyard shift are not the same. Return the sum of each previous value function and the two sums you just calculated as an integer.

Step 2: Implement the hill climbing algorithm

In a file called local_search.py implement the hill climbing search algorithm for these schedules. Try each different heuristic to see how better schedules are made with better heursitics. A state in this problem is just one instance of a schedule, and you can change the state by changing who works what shift on what day. I reccomend changing a state one shift at a time to see the algorithm work well. For hill climbing to work you want to figure out what change to the current schedule results in a better schedule according to the heuristic you are using.

Step 3: Implement simulated annealing

In the same local_search.py file, implement the simulated annealing algorithm. Again, try it with different heuristics to see what happens. Same as hill climbing, change a schedule one shift at a time.

Submitting

You need to submit three files in order for the tests to run correctly.

```
schedule.py
day.py
local_search.py
```

Grading

You will be assessed on two things: (1) implementation of the heuristics and (2) implementation of the local searches. The grading for the heuristics is cut and dry since there are tests available. The grading of your implementation will be a little more lenient, since how you choose what better states is entirely up to you. If your implementation looks like the algorithm and at leasts goes from a bad schedule to a much better schedule then you will receive a good grade.

Hints and Tips:

- In order to use the schedule class write '`from schedule import Schedule`' at the top of your file
- If you need to copy objects, use deepcopy inside python's copy module https://docs.python.org/3.5/library/copy.html

Submitted by Daniel Engbert | Assigned on 2/21/2017 2:30:00 PM | Due on 3/14/2017 11:59:00 PM

4973 Submissions Remaining

Submission Timeline

3/14/2017 1:09:13

30/30

Files Submitted

Download

day.py

local_search.py

schedule.py

day.py

```
1  #File: day.py
2  #Author: Michael Neary & Max Moraws
3  #Description: Represents a schedule
4
5  class Day:
6      def __init__(self, worker1, wor
```

Manual Grading

Name	Score
Manual Grading	0/70

Result

Name	Status	
3rd Value Function	Success	
1st Value Function	Success	
2nd Value Function	Success	

```
7           self.morning   = worker1
8           self.evening   = worker2
9           self.graveyard = worker3
10
11      #sets a shift to be a certain v
12      def set(self, shift, worker):
13          if shift == "morning":
14              self.morning = worker
15          elif shift == "evening":
16              self.evening = worker
17          elif shift == "graveyard":
18              self.graveyard = worker
19          else:
20              raise ValueError("{} is
21
22      def __repr__(self):
23          return "{},{},{}".format(se
24
```