CS 411: Artificial Intelligence I
Fall 2020
Programming Assignment 2
Due: Sunday September 27, 11:59pm

This portion of the assignment may be completed individually or in groups of 2.

In the programming portion of this assignment we will use a number of python packages to explore some of the tools we have learned about in class. Depending on your setup, this may require some installation. The following instructions were tested on a fresh installation of python 2.7.18.
On Windows, before installing the packages, you will need to install the Python Visual C++ compiler, available at `http://aka.ms/vcpython27`. Then on either Windows or Linux dowload the programming2.zip file from Blackboard and unzip it so you have access to the requirements.txt file. After that, run the following commands (in this order) to install the packages and their dependencies:

```
python -m pip install numpy==1.16.4
python -m pip install -r requirements.txt
```

Be sure to check the output of each to verify the installation occurred without error. For example, you may get an error if you failed to install the above compiler first. If you have issues installing cvxpy, see the instructions at `http://www.cvxpy.org/install/index.html`.
Once you have successfully run these, you will have installed the packages we will be using for convex programming (cvxpy), simulated annealing (simanneal), and constraint satisfaction (python-constraint).
If you have trouble (or just prefer to do so), it may be easier to set up under Anaconda. In that case,

```
conda create -n py27 python=2.7
activate py27
pip install numpy==1.16.4
pip install -r requirements.txt
```

Once you have done so point your IDE or path to python.exe in, e.g., C:/Users/Name/Anaconda/envs/py27/python.exe.
In particular, last year some Mac users ran into an issue with the C++ compiler that using Anaconda resolved.
If you are working on a machine where you lack permissions to install packages, it is also possible to install them locally. See, e.g., `https://scicomp.stackexchange.com/questions/2987/what-is-the-simplest-way-to-do-a-us`
If you run into issues getting set up, check existing threads and start a new one if needed on Piazza and we will do our best to help.
When running the autograder, the fifth problem will take some time to run, even if you haven't implemented it yet. You may with to run individual questions using the following syntax, which runs just question 1.

```
python hw2_autograder.py -q 1
```

1. **CSP: Sudoku (5 points).** We have provided most of a CSP implementation of Sudoku. You need to implement `cstAdd`, which adds the constraints. It takes a problem object (problem), a matrix of variable names (grid), a list of legal values (domains), and the side length of the inner squares (psize, which is 3 in an ordinary sudoku and 2 in the smaller version we provide as the easier test case). That is, the Sudoku puzzle consists of a psize by psize grid of psize by psize squares for psize^4 cells in total

2. **Linear Programming: Fractional Knapsack (5 points).**

   In the fractional knapsack problem you have a knapsack with a fixed weight capacity and want to fill it with valuable items so that we maximize the total value in it while ensuring the weight does not exceed the capacity c. Fractions of items are allowed. Implement the function `fractionalKnapsack` to use an LP to solve the fractional knapsack problem with 3 items of weights 5, 3, and 1 and values 2, 3, and 1 respectively for varying knapsack capacities. There is only one unit of each item available.

3. **Integer Programming: Sudoku (5 points).** For our final Sudoku, we have provided most of an IP implementation. Again, you just need to implement the constraints. Note however, unlike in the CSP version, we have not already "prefilled" the squares for you. You'll need to add those constraints yourself.

4. **Local Search: TSP (5 points).** We have provided most of a simulated annealing implementation of the famous traveling salesman problem, where you seek to visit a list of cities while minimizing the total distance traveled. You need to implement `move` and `energy`. The former is the operation for finding nearby candidate solutions while the latter evaluates how good the current candidate solution is. For this particular problem, `move` should generate a random local move without regard for whether it is beneficial. Similarly, to receive credit `energy` should calculate the total euclidean distance of the current candidate tour. There is a `distance` function you may wish to implement to help with this.

5. **Local Search: Sudoku (5 points).** Now we have the skeleton of a simulated annealing implementation of Sudoku. You need to design the `move` and `energy` functions and will receive credit based on how many of 10 runs succeed in finding a correct answer: to achieve k points $2k - 1$ runs need to pass.

Your code in the file `homework2.py` should be submitted via gradescope for evaluation. **It must be your own (or you and your partner's) code and should not be copied from any other source.** We will check for similarity to other submissions and existing resources available on the web for any cheating. Abuse of the autograder, e.g. by hardcoding solutions, is also considered cheating.

Detailed submission instructions:

1. Please submit *only* the listed files and (optionally) a README file via Gradescope

2. You can submit the files individually or submit a zip of them all, but if you zip them make sure they are in the root of the zip. (That is, make sure to zip the files themselves, not the directory that contains them.)

3. The README is to help us give you partial credit if there are any issues. Things you could comment on include anything special we need to do to run your code (e.g. if you used a non-standard package), any bugs we need to work around, and if there is a partially complete implementation which contains material you commented out for one of the questions so that the overall code will run.

4. If you worked with a partner, make sure to do it as a group submission on gradescope so that all of you get credit.

5. Please ignore any complaints Gradescope makes about the autograder.

Grading standards:

- Exceeds expectations (A): ≥ 20 points on autograder
- Meets expectations (B): $10 - 19$ points on autograder
- Partially meets expectations (C): $5 - 9$ points on autograder
- Insufficient evidence (D): ≤ 4 points on autograder