README

Timothy Eldridge*

July 29, 2015

1 Project Overview

This project implements a simple parser for the TSPLIB-95 format for traveling salesman problems (TSPs), as well as methods for calculating the length of tours and paths. In addition, two simple and similar heuristics have been implemented: the nearest neighbor algorithm and the furthest insertion algorithm.

2 Module Organization

Work is organized into five modules:

1. `parse.py`. Parses `.tsp` files of type `TSP`. Currently, this parser works only on the subset of `.tsp` files included in the directory `./tspfiles`, and has not been tested on other TSP instances in the TSPLIB library. The final destination for all parsed information is a `dict` object whose keys are `TSPLIB 95` keywords. The cities are stored in the dictionary under the key "~cities~."

2. `city.py`. Datatypes for geographical coordinates (class `GeoCoord`), cities represented as geographical points (class `GeoCity`), and cities represented as Euclidean coordinates on the plane (class `Euc_2D`). This file also contains the `distance` function, which operates on objects of classes `Euc_2D` and `GeoCity`.

3. `algorithms.py`. Simple algorithms and heuristics for calculating tours. The function `calc_in_order_tour` calculates the length of the tour

*taeldridge@ucdavis.edu

`[1,2,3,...,n,1]` for a TSP problem of dimension `n`. Also implemented in this module are the nearest neighbor and furthest insertion heuristics, respectively named `calc_nearest_neighbor_tour` and `calc_furthest_neighbor_tour`.

4. `argparser.py`. Parses command line arguments to `main.py` with the `argparse` module. Command line arguments include `-f` if the furthest insertion tour is requested, `-n` if the nearest neighbor tour is requested, and `-i` if the in-order tour length is requested.

5. `main.py`. Iterate through directories and files in order to find the `.tsp` files and print the information requested through the command line arguments.

3 Sample Invocation

The file `main.py` is intended for use as a command-line program. To get an idea of the interface, examine the help text:

```
python3 main.py --help
```

```
usage: main.py [-h] [-n] [-f] [-i] [-p] PATH [PATH ...]

Parse TSP files and calculate paths using simple algorithms.

positional arguments:
  PATH               Path to directory or .tsp file. If PATH is a directory,
                     run on all .tsp files in the directory.

optional arguments:
  -h, --help         show this help message and exit
  -n, --nearest      calculate distance traveled by nearest neighbor heuristic
  -f, --furthest     calculate distance traveled by furthest insertion
                     heuristic
  -i, --in-order     calculate the distance traveled by the in-order-tour
                     [1..n,1]
  -p, --print-tours  print explicit tours
```

4 Results

Running `main.py` on the entire director of TSP files (`./tspfiles`) is easy and pain-free:

```
python3 main.py -nfi tspfiles
```

```
TSP Problem:               a280
PATH:                      tspfiles/a280.tsp
IN-ORDER TOUR LENGTH:      2808
NEAREST NEIGHBOR LENGTH:   3157
FURTHEST NEIGHBOR LENGTH:  50172

TSP Problem:               ali535
PATH:                      tspfiles/ali535.tsp
IN-ORDER TOUR LENGTH:      3369702
NEAREST NEIGHBOR LENGTH:   253307
FURTHEST NEIGHBOR LENGTH:  4643454

TSP Problem:               berlin52
PATH:                      tspfiles/berlin52.tsp
IN-ORDER TOUR LENGTH:      22205
NEAREST NEIGHBOR LENGTH:   8980
FURTHEST NEIGHBOR LENGTH:  37742

TSP Problem:               burma14
PATH:                      tspfiles/burma14.tsp
IN-ORDER TOUR LENGTH:      4562
NEAREST NEIGHBOR LENGTH:   4048
FURTHEST NEIGHBOR LENGTH:  8854

TSP Problem:               gr137
PATH:                      tspfiles/gr137.tsp
IN-ORDER TOUR LENGTH:      97113
NEAREST NEIGHBOR LENGTH:   94124
FURTHEST NEIGHBOR LENGTH:  924837

TSP Problem:               gr202
PATH:                      tspfiles/gr202.tsp
IN-ORDER TOUR LENGTH:      58162
```

```
NEAREST NEIGHBOR LENGTH:  48524
FURTHEST NEIGHBOR LENGTH: 356085

TSP Problem:              gr229
PATH:                     tspfiles/gr229.tsp
IN-ORDER TOUR LENGTH:     179722
NEAREST NEIGHBOR LENGTH:  165928
FURTHEST NEIGHBOR LENGTH: 1959746

TSP Problem:              gr431
PATH:                     tspfiles/gr431.tsp
IN-ORDER TOUR LENGTH:     232979
NEAREST NEIGHBOR LENGTH:  212555
FURTHEST NEIGHBOR LENGTH: 3464792

TSP Problem:              gr666
PATH:                     tspfiles/gr666.tsp
IN-ORDER TOUR LENGTH:     423633
NEAREST NEIGHBOR LENGTH:  367163
FURTHEST NEIGHBOR LENGTH: 6956638

TSP Problem:              gr96
PATH:                     tspfiles/gr96.tsp
IN-ORDER TOUR LENGTH:     81015
NEAREST NEIGHBOR LENGTH:  70915
FURTHEST NEIGHBOR LENGTH: 530251

TSP Problem:              pr226
PATH:                     tspfiles/pr226.tsp
IN-ORDER TOUR LENGTH:     110417
NEAREST NEIGHBOR LENGTH:  94683
FURTHEST NEIGHBOR LENGTH: 2514865

TSP Problem:              u574
PATH:                     tspfiles/u574.tsp
IN-ORDER TOUR LENGTH:     40197
NEAREST NEIGHBOR LENGTH:  50459
FURTHEST NEIGHBOR LENGTH: 990585

TSP Problem:              ulysses16.tsp
```

```
PATH:                       tspfiles/ulysses16.tsp
IN-ORDER TOUR LENGTH:       9665
NEAREST NEIGHBOR LENGTH:    9988
FURTHEST NEIGHBOR LENGTH:   15911

TSP Problem:                ulysses22.tsp
PATH:                       tspfiles/ulysses22.tsp
IN-ORDER TOUR LENGTH:       12198
NEAREST NEIGHBOR LENGTH:    10586
FURTHEST NEIGHBOR LENGTH:   21520
```

5 Issues

Calculation of Euclidean 2-D distances does not match up with other implementations of TSP programs. The culprit is most likely the rounding function used in the `euc_2d_distance` function found in the `city` module. As per the TSPLIB '95 documentation, distances should be "round[ed] to the nearest integer (in most cases)" (6). In my reading of the TSPLIB '95 documentation, it is implied that the rounding convention used should exactly replicate the C-language `nint` function.

Nevertheless, getting Euclidean distances to match up with a previous implementation I wrote in Haskell has proven difficult. Both the Haskell `round` function and Numpy's `around` function should theoretically have the same behavior (that is, round as you learned in grade school, except for floats falling exactly equidistant from two integers – in that case, round to the nearest even integer). However, rounding errors persist in one or both of the implementations. The effect is more pronounced on TSPs with large dimension.

For example, consider the following table of in-order tour lengths calculated by this Python implementation and a previous Haskell implementation:

Problem	Python	Haskell	Percent diff. (rel. to Python)
ali535	3369702	3370175	-1.40368495 (-4)
gr666	423633	423529	2.45495511 (-4)

This Python implementation both overshoots and undershoots the Haskell implementation's calculations. As usual, the discrepancy between the two version probably originates with the particular choices each language implementation makes with floating point numbers. Making these TSP implementations match would require further study into each language's choices for floating point representations, but this project is mostly a toy, so that endeavor is ill-advised.

6 Possible Improvements

- implement the Lin-Kernighan algorithm for improving tours through programmatic permutation of city sequences
- test and patch the parser so that it operates on the full range of TSP instances provided by TSPLIB '95