CSCE 420
Programing Assignment #4
due: Tues, Oct 21 (by start of class)

Objective

The goal of this assignment is to implement Simulated Annealing in C++ and use it to solve an instance of the Traveling Salesman Problem (TSP). While the concrete objective is to find a minimal-length tour of cities in Texas, Simulated Annealing is a generic search procedure, and the code you write can be used for many different applications involving optimization.

Description of the problem

In general, TSPs take a connected graph with a set of nodes and weighted edges, and the goal is to find a minimal-length tour that visits every node exactly once. The TSP is a well-known NP-hard problem, so there is unlikely to be a tractable algorithm for finding the optimal solution. In this project, we will be finding an optimal route for visiting the 53 largest cities in Texas. This is a special version of the TSP in which the nodes are assumed to be coordinates on a 2D plane and the distances are the regular Euclidean distance. But even this restricted problem is computationally difficult.

While many people have attempted to come up with some kind of strategy (see below), Simulated Annealing (SA) turns out to be an effective and widely-used approach for solving these types of problems (i.e. for obtaining near-optimal solutions). The advantage of SA is that it does not require and special domain knowledge. It will search an arbitrary space, as long as you define an operator() function for generating successor nodes. Recall that SA is a stochastic search that generates a random neighbor and accepts it if it is better solution, or accepts it probabilistically even if it is worse. Part of the getting your implementation to work will be determining the right *temperature schedule* for the search.

Implementation

A tour can conveniently be represent as a simple list of the cities, which indicates the order in which they will be visited (don't forget the distance between the first and last cities when calculating the total length, since a formal tour has to "close the loop"). You can start the search with a random tour (permute the cities), or with a default such as alphabetical order. The state space of all possible tours is represented by all possible *permutations* of the list of cities, as long as they all appear exactly once in the list. How do you generate a successor node, or a random variant of the current tour? Given a tour, there are several ways of generating variants (many have been described in the literature). One possibility is to swap two cities in the order, or to transpose an adjacent pair. More generally, you could select a sub-segment in the middle of the order and move it to a new position, or reverse the sub-segment. Finally, you can shift (or circularly permute) the cities. Note that the latter does change the overall tour length, but it is sometimes useful to combine with other operators to make sure you are spanning the space of possible changes to generate the whole space of possible permutations.

In this assignment, you will be finding a tour that visits the top 53 cities in Texas. The cities are shown on the following map (Figure 1). If you visit the cities in alphabetical order, you zig-zag all over the state, and the total tour length is around 16,700 miles (Figure 2a). The best solution I have found to date whittles it down to around 3550 miles (Figure 2b). This took 100,000 iterations, with a temperature

schedule that started at around 30 and decreased linearly to 0. The profile of the score (tour length as a function of iteration is shown in Figure 3.

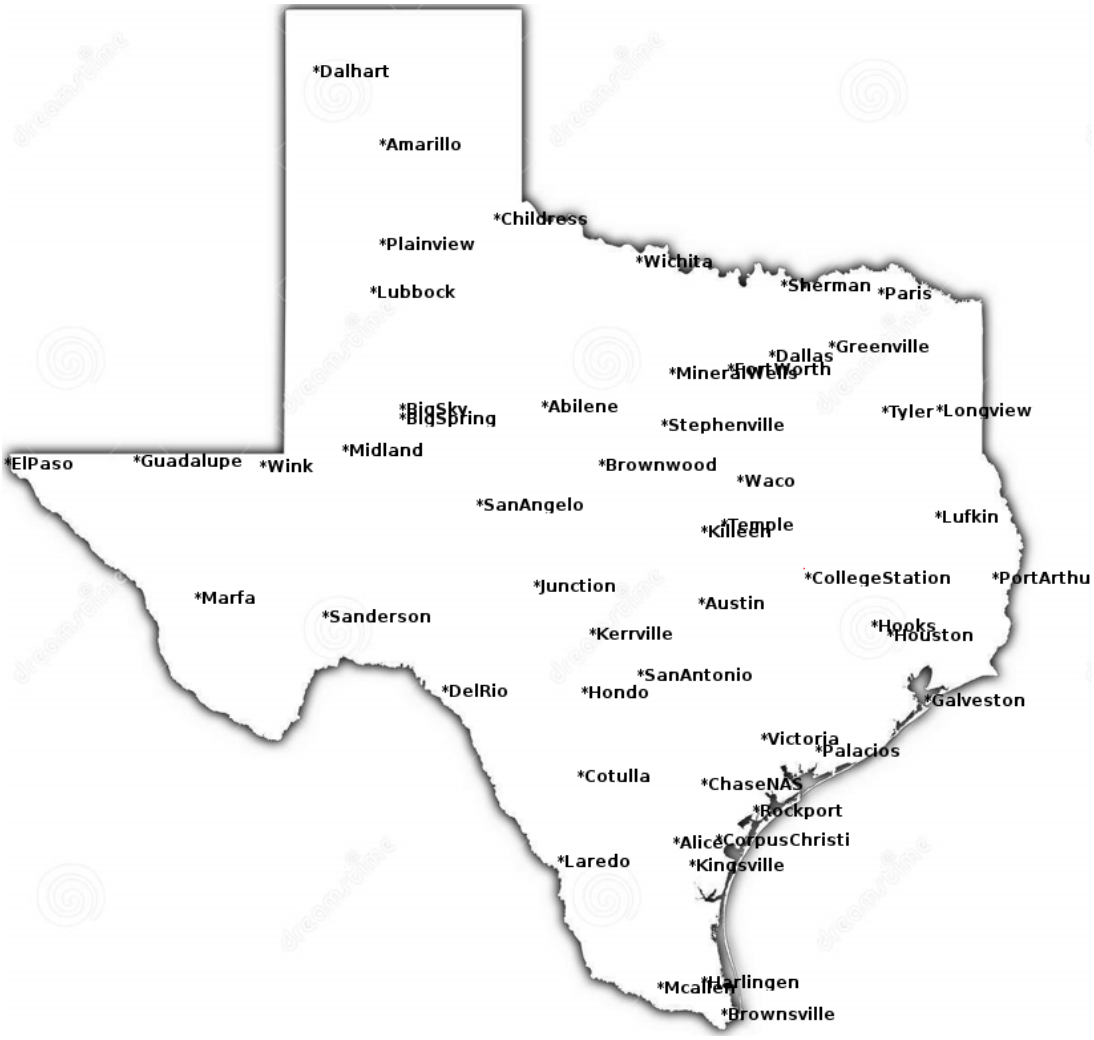

Figure 1. 53 largest cities in Texas.

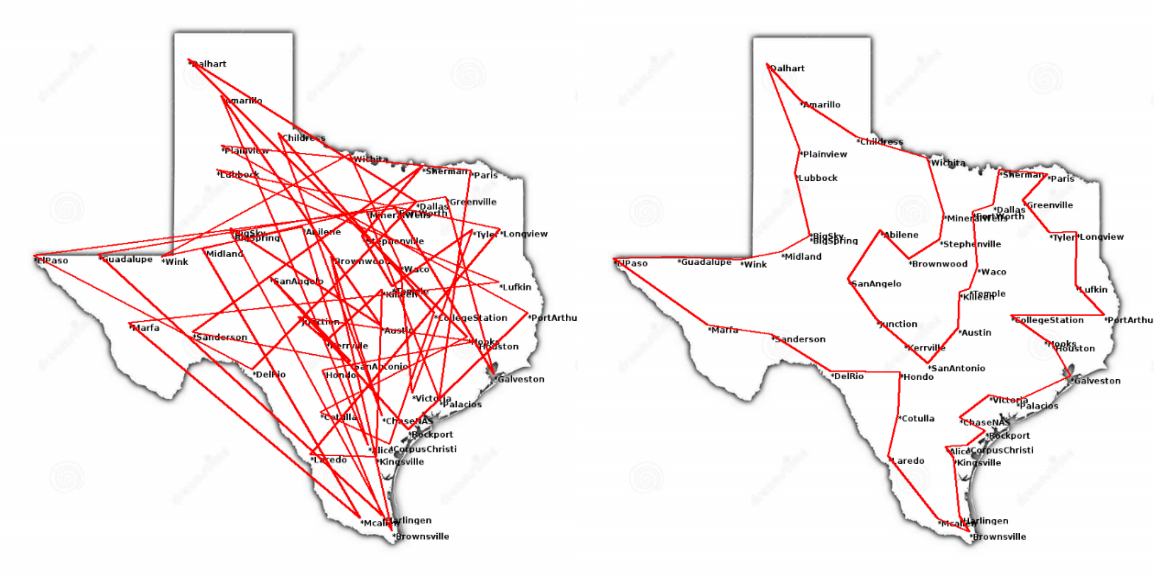

Figure 2. a) Tour in alphabetical order, length=16,697 miles. b) One of the best tours found, length=3548 miles.

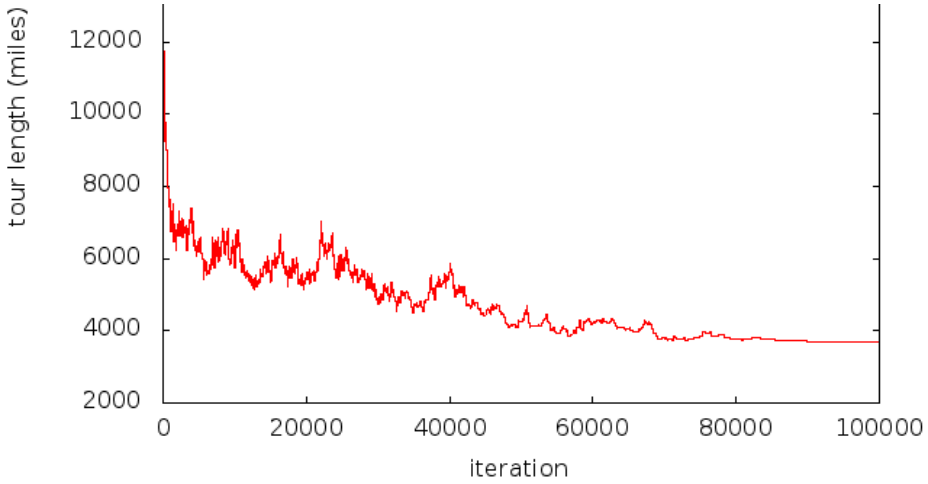

Figure 3. Profile showing how quality of solution (tour length) decreased over 100,000 iterations, converging at a tour of length 3679, using a starting temperature of 60.

The input file (which can be downloaded from the course website) consists of the city name (no spaces), followed by the latitude and longitude coordinates (based on airports). For example:

```
Abilene              32.42  99.68
Alice                27.73  98.03
Amarillo             35.23 101.70
...
```

In my code, I found it useful to print out some *tracing information* with each iteration (shown in the transcript below), and you should do the same. At each iteration, I print the the length of the current tour and the new variant (successor) generated, the difference in scores, and the probability of accepting a worse solution. Then, if the successor is accepted, I print out the new tour (list of city names). An example transcript is shown at the end. In the first few rounds, modified tours are accepted on nearly every iteration. Toward the end, after it has converged, updates become much less frequent. I highted parts of the tours that changed in the first few iterations.

For your convenience, I have implemented a web-based mapping tool for visualizing tours. It is at `http://orca2.tamu.edu/tom/TourOfTexas.html` It takes a list of cities and displays the tour for you, along with calculating the total tour length at the bottom. This is what I used to generate the figures above.

Calculating Distances in Miles from Latitude/Longitude Coordinates

One technical detail is how to calculate distances in miles between cities from latitude/longitude coordinates. I could have defined the input data file to contain all pairwise distances, but there would be a lot to read in (53*52/2=1378 pairs). But this is unnecessary if you have the geo-coordinates of each city. Calculating distance between two points on the globe from their lat/long coordinates is non-trivial. Remember that the distance between longitudinal lines changes depending on latitude (how far from the equator you are) and shrinks as you approach the poles. You can find a description of the formulas for calculating the distance in miles using the Haversine method on these pages:

```
http://www.ig.utexas.edu/outreach/googleearth/latlong.html

http://andrew.hedges.name/experiments/haversine/

dlon = lon2 - lon1

dlat = lat2 - lat1

a = (sin(dlat/2))^2 + cos(lat1) * cos(lat2) * (sin(dlon/2))^2

c = 2 * atan2( sqrt(a), sqrt(1-a) )

d = R * c (where R is the radius of the Earth, 3961 miles)
```

Note, you must convert latitude and longitude values from degrees to radians for these formulas!

```
rad = deg*3.14159/180.0
```

Greedy Strategies

You might be tempted to think that there must be any easier strategy for solving TSPs such as this, like by exploiting nearest neighbors in the map. Many people have tried, and there are some effective approximation algorithms for TSP based on this idea. However, it is still difficult to find good solutions, and these methods do not produce tours as good as the ones found by SA. Here are two examples. First, I tried greedy strategy 1 where I started with Abilene, then picked the closest city to it

(Brownwood), then the closest remaining city to it (Stephenville), and so on. This produced the tour shown in Figure 4a, which is clearly sub-optimal and has a total length of 4319 miles. When you get down to Brownsville, the closest next choice is to jump all the way up to Galveston. Next I tried greedy strategy 2, in which I started by joining the 2 closest cities (BigSky and BigSpring), and then the next 2 closest, until they were all joined up (forcing no branches or cycles to be created). This produced the tour shown in Figure 4b, which has a length of 4032 miles. By comparison, the tours found by Simulated Annealing are routinely around 3500 miles, which is 15-20% shorter. (I do not actually know what the true minimum tour length is for this problem.)

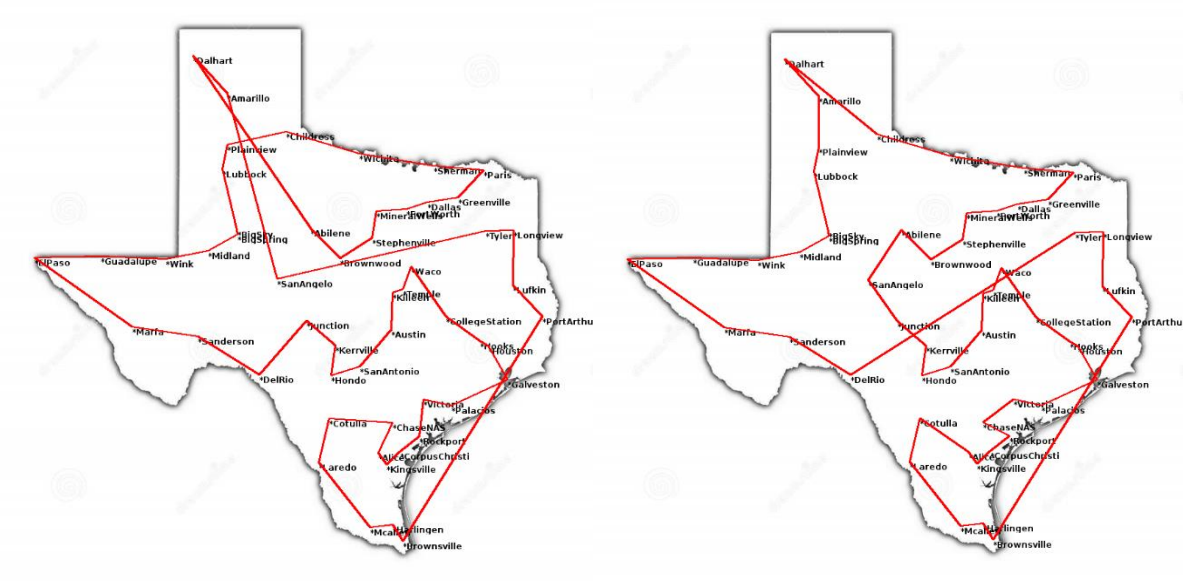

Figure 4. Tours produced by 2 greedy strategies.

What to Turn in

- You will submit your code for testing using the web-based CSCE *turnin* facility, which is described here: https://wiki.cse.tamu.edu/index.php/Turning_in_Assignments_on_CSNet
- Include a brief document that describes how to compile and run your code, and an example of the output (the best tour you could find, the temperature schedule used, and the number of iterations required).
- Also turn in a plot of the decrease in tour length over the iterations, like Figure 3 above.

Example Run

```
>TSP 100000 60
initial state, tour length=16696.9 miles
Abilene Alice Amarillo Austin BigSky BigSpring Brownsville Brownwood ChaseNAS
Childress CollegeStation CorpusChristi Cotulla Hooks Dalhart Dallas DelRio ElPaso
FortWorth Galveston Greenville Guadalupe Harlingen Hondo Houston Junction Kerrville
Killeen Kingsville Laredo Longview Lubbock Lufkin Marfa Mcallen Midland MineralWells
Palacios Paris Plainview PortArthur Rockport SanAngelo SanAntonio Sanderson Sherman
Stephenville Temple Tyler Victoria Waco Wichita Wink

iter=0 len=16696.9 newlen=16460.5 delta=-236.402 temp=60 p<51.4206
update! len=16460.5
Abilene Alice Amarillo Austin BigSky BigSpring Brownsville Brownwood ChaseNAS
Childress CollegeStation CorpusChristi Cotulla Hooks Dalhart Dallas FortWorth ElPaso
DelRio Galveston Greenville Guadalupe Harlingen Hondo Houston Junction Kerrville
Killeen Kingsville Laredo Longview Lubbock Lufkin Marfa Mcallen Midland MineralWells
Palacios Paris Plainview PortArthur Rockport SanAngelo SanAntonio Sanderson Sherman
Stephenville Temple Tyler Victoria Waco Wichita Wink

iter=1 len=16460.5 newlen=16268.5 delta=-192.038 temp=59.9994 p<24.5489
update! len=16268.5
Abilene Alice Amarillo Austin BigSky BigSpring Brownsville Brownwood ChaseNAS
Childress CollegeStation CorpusChristi Cotulla Hooks Dalhart Dallas FortWorth ElPaso
DelRio Galveston Greenville Guadalupe Harlingen Hondo Houston PortArthur Plainview
Paris Palacios MineralWells Midland Mcallen Marfa Lufkin Lubbock Longview Laredo
Kingsville Killeen Kerrville Junction Rockport SanAngelo SanAntonio Sanderson Sherman
Stephenville Temple Tyler Victoria Waco Wichita Wink

iter=2 len=16268.5 newlen=16700 delta=431.524 temp=59.9988 p<0.000752419
iter=3 len=16268.5 newlen=16479.3 delta=210.786 temp=59.9982 p<0.0298012
iter=4 len=16268.5 newlen=16260.7 delta=-7.73926 temp=59.9976 p<1.13768
update! len=16260.7
BigSky Abilene Alice Amarillo Austin BigSpring Brownsville Brownwood ChaseNAS
Childress CollegeStation CorpusChristi Cotulla Hooks Dalhart Dallas FortWorth ElPaso
DelRio Galveston Greenville Guadalupe Harlingen Hondo Houston PortArthur Plainview
Paris Palacios MineralWells Midland Mcallen Marfa Lufkin Lubbock Longview Laredo
Kingsville Killeen Kerrville Junction Rockport SanAngelo SanAntonio Sanderson Sherman
Stephenville Temple Tyler Victoria Waco Wichita Wink
...
...
iter=99990 len=3679.18 newlen=3679.18 delta=0 temp=0.006 p<1
update! len=3679.18
Wink Guadalupe ElPaso Marfa Sanderson DelRio Junction Kerrville SanAntonio Hondo
Cotulla Laredo Mcallen Brownsville Harlingen Kingsville Alice CorpusChristi ChaseNAS
Rockport Palacios Victoria Austin CollegeStation Hooks Houston Galveston PortArthur
Lufkin Longview Tyler Waco Temple Killeen Brownwood Abilene Stephenville MineralWells
FortWorth Dallas Greenville Paris Sherman Wichita Childress Amarillo Dalhart Plainview
Lubbock BigSky BigSpring SanAngelo Midland

iter=99991 len=3679.18 newlen=3770.64 delta=91.4556 temp=0.0054 p<0
iter=99992 len=3679.18 newlen=4651.94 delta=972.759 temp=0.0048 p<0
iter=99993 len=3679.18 newlen=4179.91 delta=500.729 temp=0.0042 p<0
iter=99994 len=3679.18 newlen=4498.68 delta=819.504 temp=0.0036 p<0
iter=99995 len=3679.18 newlen=4527.83 delta=848.647 temp=0.003 p<0
iter=99996 len=3679.18 newlen=4262.32 delta=583.138 temp=0.0024 p<0
iter=99997 len=3679.18 newlen=4363.94 delta=684.761 temp=0.0018 p<0
iter=99998 len=3679.18 newlen=4490.86 delta=811.678 temp=0.0012 p<0
iter=99999 len=3679.18 newlen=4609.07 delta=929.887 temp=0.0006 p<0
```