SIMULATED ANNEALING FOR TRAVELING SALESMAN PROBLEM

ARAVIND SESHADRI

1. Traveling Salesman Problem

Traveling Salesman Problem (TSP) has been an interesting problem for a long time in classical optimization techniques which are based on linear and nonlinear programming. TSP can be described as follows: Given a number of cities to visit and their distances from all other cities know, an optimal travel route has to be found so that each city is visited one and only once with the least possible distance traveled. This is a simple problem with handful of cities but becomes complicated as the number increases.

2. Simulated Annealing

There are several well know classical methods for finding a near optimal solution using linear and nonlinear programming for TSP's. Simulated Annealing is not based on any of the classical optimization techniques but is based on heuristics from annealing process. Based on the motivation from how crystalline structures are formed from annealing process, a heuristic method called simulated annealing was developed and had a lot of interesting results. I have not put the effort to explain everything in detail about simulated annealing but a quick google on simulated annealing would get you started.

To make the long story short, simulated annealing is similar to hill climbing or gradient search with a few modifications. In gradient based search the search direction is dependent on gradient and hence the function to be optimized should be a continuous function without discontinuities. Simulated Annealing does not require the function to be smooth and continuous since it is not based on gradient of the function. Consider a minimization problem with a lot of local optima. During the initial search phase (when temperature is high) local search[1] is carried out and routes which minimize the total distance are always accepted. To escape the problem of getting stuck in a local minima occasionally routes with distance more than the current route is also accepted but with a probability similar to the probability in the dynamics of the annealing process. As the temperature decreases, this probability of accepting a bad solution is decreased and in the final stages it becomes similar to gradient based search. This idea is blend between a completely random search and a gradient based search with some heuristics based on the annealing process.

[1]Search meaning random routes are made and their corresponding distances calculated.

3. 20 Cities TSP

A generic simulated annealing for 20 cities traveling salesman problem from TSPLIB[2] archive. The best solution found is given in figure 1 and the total optimal distance to travel 20 cities is 4.030656 units.

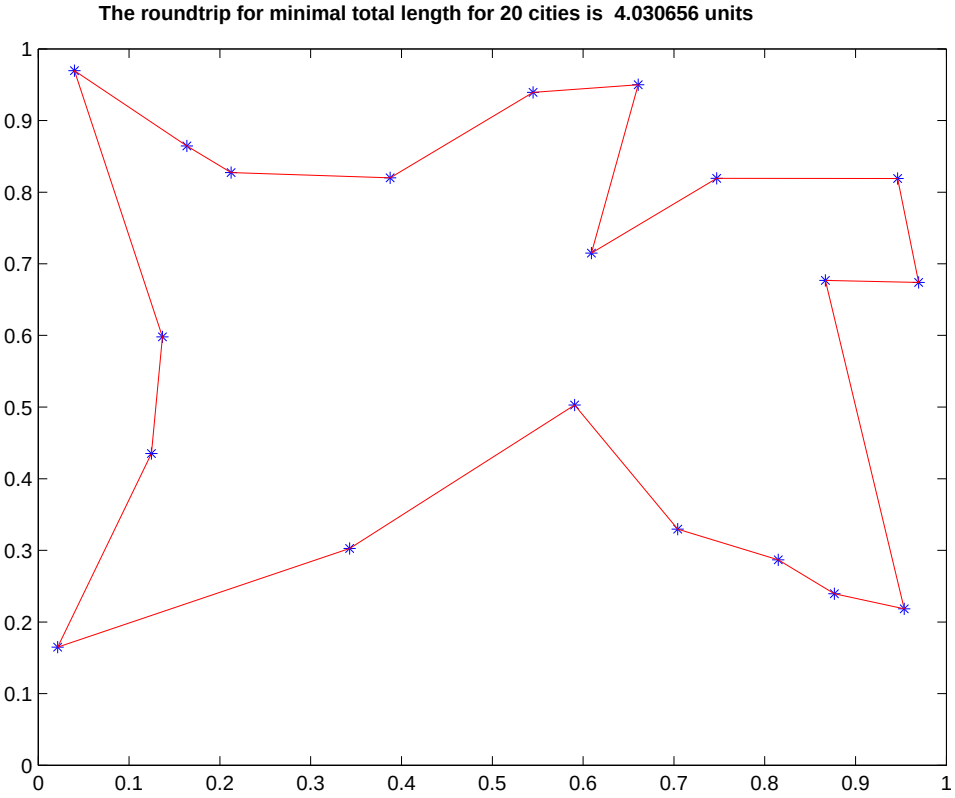

FIGURE 1. The Best Solution

3.1. **Algorithm.** The algorithm for simulated annealing is given below. The MATLAB source code is provided below. Feel free to edit the code and make changes. You can use this code to run any of the problem in the TSPLIB as long as you follow the procedure for loading the corresponding data file as described in the next subsection. I will have to warn you that running a large data set will take a lot of time and requires a lot of memory and processing. I am only trying to get you started on simulated annealing and I am sure my code is not optimized for large data set. I would be really happy if someone is willing to help me out in optimizing my code as well as in refining my algorithm.

3.1.1. *Load Data File.* This is an example for loading the data file.

```
function d = datafile()
cities =
```

[2]http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

```
[0.6606,0.9695,0.5906,0.2124,0.0398,0.1367,0.9536,0.6091,0.8767,...
0.8148,0.3876,0.7041,0.0213,0.3429,0.7471,0.5449,0.9464,0.1247,0.1636,...
0.8668,;0.9500,0.6740,0.5029,0.8274,0.9697,0.5979,0.2184,0.7148,0.2395,...
0.2867,0.8200,0.3296,0.1649,0.3025,0.8192,0.9392,0.8191,0.4351,0.8646,...
0.6768];
save cities.mat cities -V6;
```

When `datafile` function is called an array containing the coordinates of the cities are loaded into a file called `cities.mat`. The file is stored in structure format. You can write your own datafile function by visiting

www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp

and you can use any data file in the web site. An example is shown below.

```
function c = loadeil101()
% LOADEIL101 Loads the data file.
% COMMENT : 101-city problem (Christofides/Eilon)
% TYPE : TSP
% DIMENSION : 101
% Source: www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp
clear all; temp_x = [1 41 49 2 35 17 3 55 45 . . . . . 98 19 21 99
20 26 100 18 18 101 35 35]; cities = [temp_x(:,2)';temp_x(:,3)'];
save cities.mat cities -V6;
```

The first column of the data file from TSPLIB is usually the city number and the next couple of columns represent the coordinates for the cities. My function looks for file named as `cities.mat` in a structure format which when loaded to the workspace has the variable name cities. I sure there is an elegant way to make everything completely independent but I am lazy to modify anything. Hence make sure the data file has the last couple of lines the same unless you know what you are doing.

3.1.2. *Function for calculating the total distance.*

```
function d = distance(inputcities)
% DISTANCE
% d = DISTANCE(inputcities) calculates the distance between n cities as
% required in a Traveling Salesman Problem. The input argument has two rows
% and n columns, where n is the number of cities and each column represent
% the coordinate of the corresponding city.

d = 0;
for n = 1 : length(inputcities)
    if n == length(inputcities)
        d = d + norm(inputcities(:,n) - inputcities(:,1));
    else
        d = d + norm(inputcities(:,n) - inputcities(:,n+1));
    end
end
```

The cost function for a symmetric traveling salesman problem is the distance between the cities. This function calculates the distance been `n` cities when `distance(n)` function is called.

3.1.3. *Function to plot the route.*

```
function f = plotcities(inputcities)
% PLOTCITIES
% PLOTCITIES(inputcities) plots the location of cities from the argument
% inputcities. Inputcities argument has 2 rows and n columns, where n is
% the number of cities. Apart from the location the symmetric route is also
% plotted.
shg
temp_1  = plot(inputcities(1,:),inputcities(2,:),'b*');
set(temp_1,'erasemode','none');
temp_2 = line(inputcities(1,:),inputcities(2,:),'Marker','*');
set(temp_2,'color','r');
x = [inputcities(1,1) inputcities(1,length(inputcities))];
y = [inputcities(2,1) inputcities(2,length(inputcities))];
x1 = 10*round(max(inputcities(1,:))/10);
y1 = 10*round(max(inputcities(2,:))/10);
if x1 == 0
    x1 = 1;
end if y1 == 0
    y1 = 1;
end
axis([0 x1 0 y1]);
temp_3 = line(x,y);
set(temp_3,'color','r');
dist = distance(inputcities);
distance_print = sprintf(...
     'The roundtrip length for %d cities is % 4.6f units'...
     ,length(inputcities),dist);
text(x1/15,1.05*y1,distance_print,'fontweight','bold');
drawnow;
```

3.1.4. *Function to swap cities.*

```
    function s = swapcities(inputcities,n)
    % SWAPCITIES
    % s = SWAPCITIES(inputcities,n) returns a set of m cities where n cities
    % are randomly swaped.
    s = inputcities; for i = 1 : n
        city_1 = round(length(inputcities)*rand(1));
        if city_1 < 1
            city_1 = 1;
        end
        city_2 = round(length(inputcities)*rand(1));
        if city_2 < 1
            city_2 = 1;
        end
        temp = s(:,city_1);
        s(:,city_1) = s(:,city_2);
        s(:,city_2) = temp;
```

```
end
```

3.1.5. *Function to perform Simulated Annealing.*

```
function s = simulatedannealing(inputcities,initial_temperature,...
    cooling_rate,threshold,numberofcitiestoswap)
% SIMULATEDANNEALING
% S = SIMULATEDANNEALING(inputcities,initial_temperature,cooling_rate)
% returns the new configuration of cities with an optimal solution for the
% traveling salesman problem for n cities.
%
%The input arguments are
% INPUTCITIES         - The cordinates for n cities are represented as 2
%                       rows and n columns and is passed as an argument for
%                       SIMULATEDANNEALING.
% INITIAL_TEMPERATURE - The initial temperature to start the
%                       simulatedannealing process.
% COOLING_RATE        - Cooling rate for the simulatedannealing process.
%                       Cooling rate should always be less than one.
% THRESHOLD           - Threshold is the stopping criteria and it is the
%                       acceptable distance for n cities.
% NUMBEROFCITIESTOSWAP- Specify the maximum number of pair of cities to
%                       swap. As temperature decreases the number of cities
%                       to be swapped decreases and eventually reaches one
%                       pair of cities.

global iterations;
temperature = initial_temperature;
initial_cities_to_swap = numberofcitiestoswap;
iterations = 1;
complete_temperature_iterations = 0;
while iterations < threshold
    previous_distance = distance(inputcities);
    temp_cities = swapcities(inputcities,numberofcitiestoswap);
    current_distance = distance(temp_cities);
    diff = abs(current_distance - previous_distance);
    if current_distance < previous_distance
        inputcities = temp_cities;
        plotcities(inputcities);
        if complete_temperature_iterations >= 10
            temperature = cooling_rate*temperature;
            complete_temperature_iterations = 0;
        end
        numberofcitiestoswap = round(numberofcitiestoswap...
            *exp(-diff/(iterations*temperature)));
        if numberofcitiestoswap == 0
            numberofcitiestoswap = 1;
        end
        iterations = iterations + 1;
        complete_temperature_iterations = complete_temperature_iterations + 1;
```

```
        else
            if rand(1) < exp(-diff/(temperature))
                inputcities = temp_cities;
                plotcities(inputcities);
                numberofcitiestoswap = round(numberofcitiestoswap...
                    *exp(-diff/(iterations*temperature)));
                if numberofcitiestoswap == 0
                    numberofcitiestoswap = 1;
                end
                complete_temperature_iterations = complete_temperature_iterations + 1;
                iterations = iterations + 1;
            end
        end
        clc
        fprintf('\t\t\tIterations = %d\n',iterations);
        fprintf('\t\t\tTemperature = %3.8f\n',temperature);
        fprintf('\t\t\tIn current temperature for %d times\n',...
            complete_temperature_iterations);
    end
    previous_distance
```

3.1.6. *The GUI to execute Simulated Annealing.* A GUI is created for experimenting with different data set, with different operating conditions. Figure 2 shows the GUI. rate as 0.9 and the initial temperature as 20 units. In the GUI play around

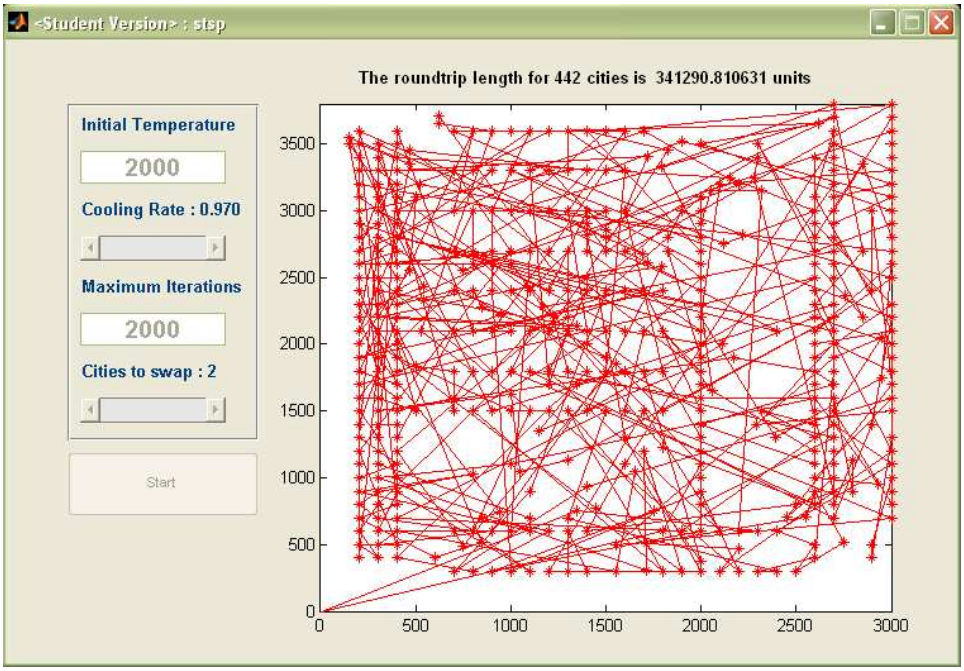

FIGURE 2. A Graphics User Interface for Traveling Salesman Problem using Simulated Annealing.

with different initial temperatures, cooling rates and cities to swap. There is no best recipe which can solve every problem hence you might have to play around a little bit. A detailed explanation of `simulatedannealing` function will be presented in the next section.

4. Simulated Annealing Algorithm

A sample optimal solution for 101 cities is shown in figure 3. Figures 4 - 7 show

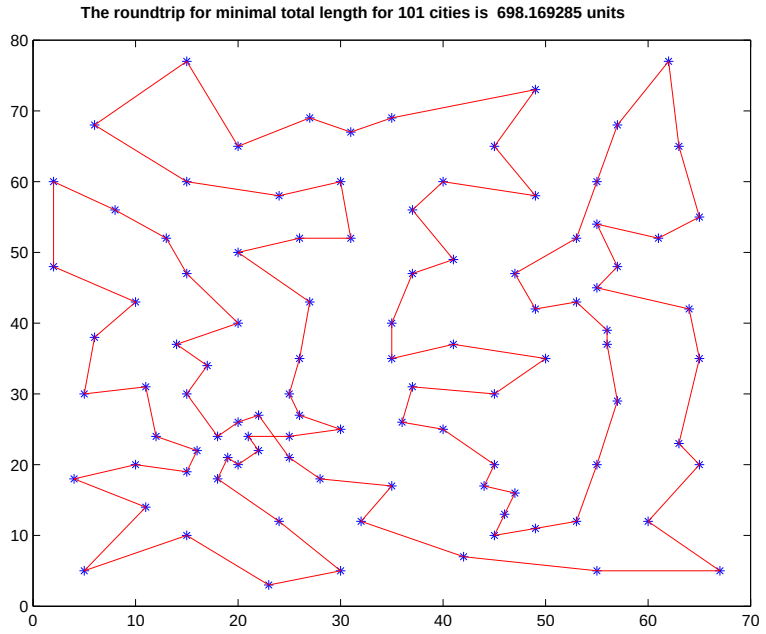

FIGURE 3. Optimal Solution for 101 cities TSP

a few other examples. These figures does not represent the final optimal solution since the simulations were stopped in the middle.

4.1. *simulatedannealing()*.

The `simulatedannealing()` function has five input arguments.

- First argument - Array of cities
- Second argument - Initial temperature
- Third argument - Cooling rate
- Fourth argument - Threshold, in this case the threshold is the total number of iterations. Threshold can also be sent in terms of optimal solution or in terms of final temperature but for the algorithm mentioned it is set in terms of iterations.
- Fifth argument - Number of cities to swap. Cities are swapped each time to get a random new configuration. From the current configuration a pair of cities may be swapped at random or any number of pairs of cities may be interchanged to get the random configuration. In this algorithm, as the temperature decreases the number cities to be swapped also decreases and

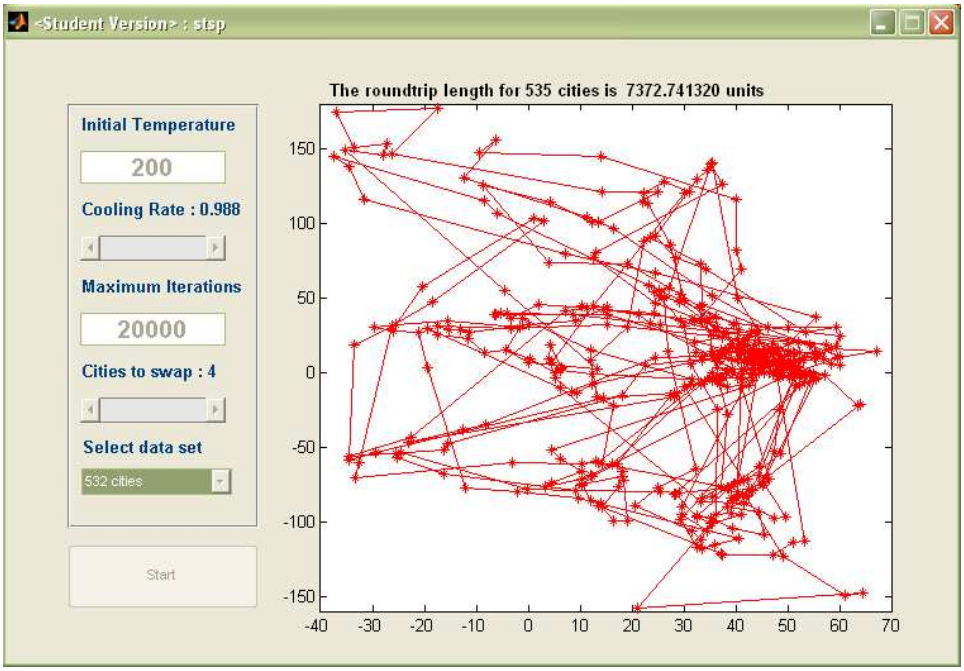

FIGURE 4

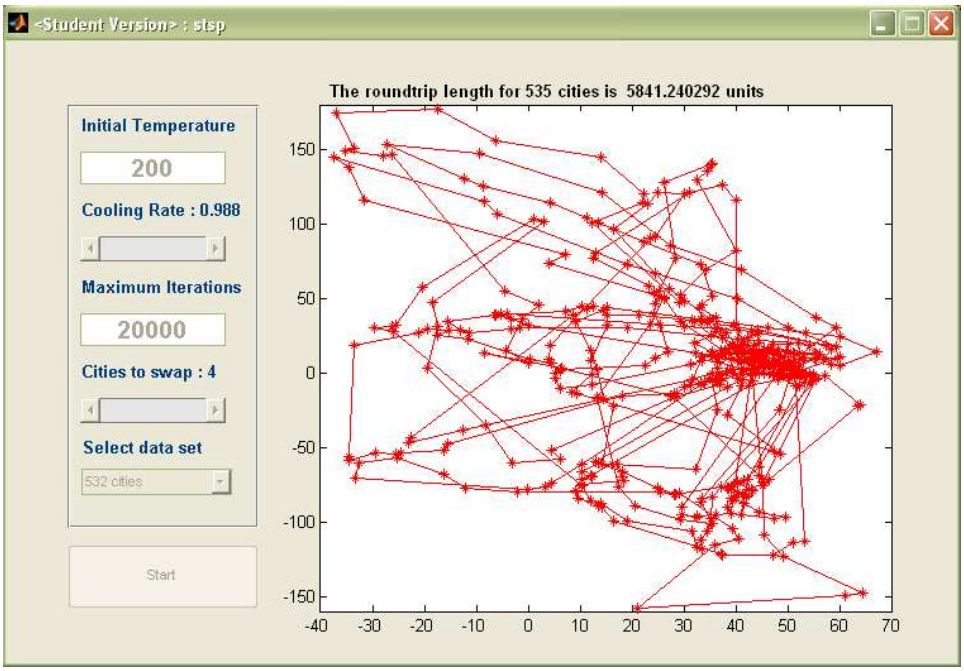

FIGURE 5

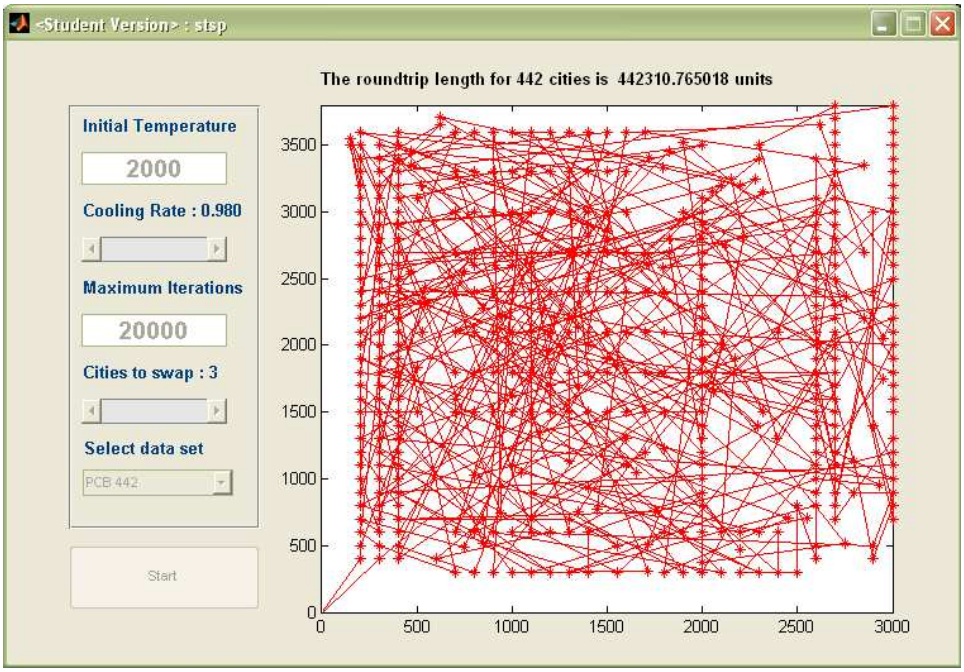

FIGURE 6

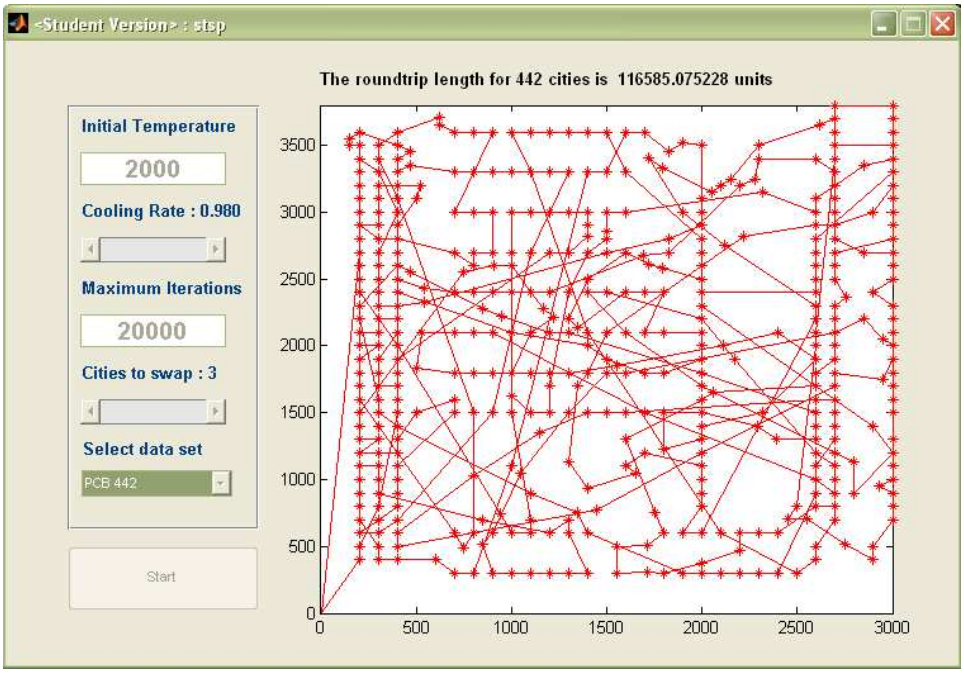

FIGURE 7

finally only one pair of cities are swapped. The rate at which this number is reduced is given by (4.1).

$$c = \text{round}\left(x * e^{-\frac{\triangle E}{T}}\right) \tag{4.1}$$

where, c is the new number of pair of cities to be swept, x is the current number of pair of cities swapped, $\triangle E$ is the difference in current and previous cost function (total distance) and T is the current temperature.

This function is run as long as the threshold is not met. Also the temperature is not decreased after each iteration as proposed by [5], where the decrease in temperature is very slow as given in (4.2).

$$t = \frac{t}{1 + \beta t} \tag{4.2}$$

where, β is sufficiently small value and is determined based on the problem. The cooling schedule is carried out geometrically as given by (4.3)

$$t = \alpha t \tag{4.3}$$

where $\alpha < 1$. To make the algorithm realistic, search is carried out at each temperature for a given number of iterations before reducing the temperature. Also at low temperatures the search is carried out for a longer time before reducing the temperature. This can be done by simply making the temperature decrease only when $\triangle E < 0$, while allowing uphill moves without temperature change.

Another key feature is the probability of acceptance when a new configuration has higher cost function value than the current configuration. Instead of using (4.4), (4.5) is employed.

$$P(A) = e^{\left(-\frac{\triangle E}{kT}\right)} \tag{4.4}$$

$$P(A) = e^{\left(-\frac{\triangle E}{T}\right)} \tag{4.5}$$

where, k is the number of iterations. This improved the performance of the simulated annealing algorithm when a large TSP problem is considered.

5. Potential Problematic Issues

5.1. **Computation Time.** One of the main drawbacks for simulated annealing is the computation time. For better results the cooling has to be carried out very slowly and this significantly increases the computation time. Clearly when moving from 20 cities to 101 cities a significant rise in computation time is observed. Various computations like, computation of the cost function, computation of probability of acceptance etc. increases the computation time when the dimension of the problem grows.

5.2. **Probability of Acceptance.** Occasionally if temperature is high and if the cost function is not reduced much from the initial configuration, it has been observed in my simulations that, probability stays very high since the iterations are high when (4.4) is used. This causes the simulation to run for a very long time.

5.3. **Local Minima.** Simulated annealing has been applied to very complex problems with various constraints. The complexity of the structures is due to numerous local minima whose number are often exponential in the numbers of degrees of freedom [3].Occasionally the solution is trapped in a local minima even when the cooling is very slow. And when the probability of acceptance is very low, it becomes difficult to come out of the local minima as observed in figure 3. This would mean to restart simulated annealing from the beginning.

5.4. **Cooling Schedule.** Cooling schedule is very important to simulate annealing. Each problem requires a unique cooling schedule and it becomes very difficult to pick the most appropriate schedule within a few simulations. A fast cooling schedule is similar to greedy algorithm while a very slow cooling schedule will require a lot of computation. An optimal cooling schedule is one which can reach the global minimum with minimum amount of computation. Obtaining an optimal cooling schedule is a research field by itself.

5.5. **Good Set of Performance Parameters.** Simulated annealing is a stochastic search with some heuristics. Hence choosing an optimal set of performance parameters becomes difficult and comes with experience. Good set of parameters dictate the convergence to global minima in the least amount of time. As noted by [2], it is difficult to find an optimal parameter set and there is no fixed strategy to find such a parameter set.

6. Remarks

6.1. **Computation Time.** One of the main drawback of simulated annealing is computation time. Several suggestions are proposed by Graham [3] which are

- **Approximating the cost function:** Cost functions are calculated each iteration and hence a simplified evaluation of cost function is necessary to reduce the computation time. A simplified evaluation of cost function which approximates the function while at the same time provides a good indication as to the quality of the solution is therefore chosen.
- **Evaluation of probability of acceptance:** The exponential calculation in determining the probability of acceptance takes a lot of computation resources and it has been shown by [6] that the acceptance calculation take one third of the computation time. A better was of calculating the acceptance criteria has been shown to be approximating, $P(A) = 1 - \frac{\triangle E}{t}$. Another way to reduce computation time may be by generating a look-up table for a set of values for $\frac{\triangle E}{t}$ and the rounded integer value of $\frac{\triangle E}{t}$ was used to access the look-up table.

[3] http://www.cs.nott.ac.uk/ gxk/aim/notes/simulatedannealing.doc. I am new to LaTeXand I am not able to figure out how to get a tilde in front of `gxk` in the URL. While looking for the website make sure to add tilde in front of `gxk`

- **Retrieving solutions from cache:** Occasionally some solutions (partial or complete) are stored in the memory and when the solutions are to be evaluated they are retrieved from the memory rather than evaluating them. This saves considerable computation time but needs a lot of memory.

6.2. **Probability of Acceptance.** To over come the problem observed in 5.2, equation (4.4) is modified to (4.5), so that the effect of number of iterations does not increase the probability of acceptance. With a slow decrease of temperature, (4.5) provides a better convergence. A better definition for k, the constant which is similar to Boltzman constant, has to be made.

6.3. **Local Minima.** It is common to start simulated annealing from a random configuration. The performance of simulated annealing may be improved if more information is know about the problem in hand. Hence it might be better to start from a configuration which is a local minima, like a configuration obtained by a greedy algorithm search. Starting from a local minimal with initial high temperature will provide an opportunity to escape the local minima and attain a better solution, possibly a global minima.

6.4. **Hybridization.** Hybridization refers to combining two search algorithms to solve a given problem. It is also referred to as memetic algorithms [4]. This is often a population based search such as genetic algorithm with local searches performed by other algorithms like simulated annealing, greedy algorithm etc. A similar hybridization may be carried out with simulated annealing

6.5. **Initial Temperature.** If the initial temperature is too high then it simulates a random search since most of the random configurations are accepted because the probability of acceptance is high. This means that until the temperature cools down sufficiently, the search is nothing but a random search and it is similar to starting an algorithm form a random configuration after a particular temperature is reached. While a low temperature starting point is much like a greedy algorithm, preventing the escape of local minima. Hence choosing an initial temperature requires experience and insight into the problem in hand.

6.6. **Cooling Schedule.** Cooling schedule is the heart of simulated annealing. As seen in 6.5 most of the optimization is done in the middle stages of cooling schedule. In [2] it is shown that there exists a fixed temperature at which the performance of an annealing scheme is optimum and if search is carried out for a longer time at the temperature, an optimal solution can be obtained. A cooling schedule similar to one proposed in [1] is used where the search is carried out for a certain time at each temperature before reducing the temperature. After 'n' number of randomly accepted configurations, the temperature is cooled geometrically. This modification significantly improved the performance of annealing algorithm when implemented in my algorithm. Some interesting conclusions of [2] include

- Simulated annealing is an extremely efficient heuristic for Quadratic Assignment Problem (QAP).
- For the QAP, a sequential generation of neighbors is superior to a random selection method in an annealing scheme.
- There exists a fixed temperature at which the performance of an annealing scheme is optimized.

6.7. **Stopping Criteria.** Stopping criteria is also an issue in simulated annealing. Various guidelines are provided for enforcing a stopping criteria and most of them are dependent on problem in hand. Some of the typical criteria's include

- Maximum number of iterations reached.
- No change in the current configuration for a very long time.
- The temperature is very low or the frozen state is reached, where no possibility of uphill move and no change in the cost function is observed.

It is common to store the 'Best so Far' (BSF) solution, so that when ever an annealing process is stopped that configurations can be retrieved. It is possible that simulated annealing search might have moved from a global minima during the initial stages of cooling and hence to avoid such a situation BSF solution is constantly stored in the memory.

7. Concluding Remarks

Simulated annealing has a potential of finding an optimal solution provided enough time is given for the annealing process. Indeed a simple concept of annealing has significant effect on finding an optimal solution, thus simulated annealing is indeed a good motivation for other heuristic algorithms. I have not used any of the *'n-opt'* methods, but if time permits I will try to update this algorithm with those methods.

Thankyou for taking your time in reading this article. I am sure there would be a lot of bugs and mistakes in this program. I greatly welcome all the comments, suggestions and improvements to this program. Kindly email me at aravind.seshadri@okstate.edu to send you comments and suggestion.

References

1. R. E. Burkard and F. Rendl, *A thermodynamically motivated simulation procedure for combinatorial optimization problems*, European Journal of Operational Research **17** (1984), no. 2, 169 174.
2. David T. Connolly, *An improved annealing scheme for the qap*, European Journal of Operational Research **46** (1990), no. 1, 93 – 100.
3. Karl Heinz Hoffmann and Peter Salamon, *The optimal simulated annealing schedule for a simple model*, J. Phys. A: Math. Gen. **23** (1990), 3511 – 3523.
4. Graham Kendall, *Simulated annealing*, www.cs.nott.ac.uk/ gxk/aim/notes/simulatedannealing.doc, 2005.
5. M Lundy and A Mees, *Convergence of an annealing algorithm*, Mathematical Programming: Series A and B **34** (1986), no. 1, 111 – 124.
6. Lyle A McGeoch, David S Johnson, Cecilia R Aragon, and Catherine Schevon, *Optimization by simulated annealing: An experimental evaluation part ii, graph coloring and number partitioning*, Operations Research **39** (1991), 378 – 406.

E-mail address: `aravind.seshadri@okstate.edu`