Report of Mini-Project

Apply Simulated Annealing to Traveling Salesman Problem

Statement of the Problem

Traveling Salesman Problem (TSP) is a famous problem in algorithm, which asks for shortest route that passes all cities once and only once, and finishes up at starting city. Where the length of route and distance traveled is calculated as Euclidean distance between the coordinates of the cities.

TSP is a problem with exponential complexity, and proven to be NP-complete. By applying iterative improvement algorithms, we may be able to find solution to this problem in a more systematic and efficient way. The challenge of this approach is to reach global optimal without being restricted to local optimal.

Because exhaust search is not a viable solution due to the exponential complexity, it is attempted to apply simulated annealing to find (sub-optimal) solutions to TSP with 100-200 cities randomly distributed in a Euclidean space. The proposal of the project is to understand how fast and effective simulated annealing algorithm can generate a solution to TSP problem.

Approach

This project will mainly apply simulated annealing algorithm to solve TSP. Simulated annealing is an evolitional algorithm that takes random move to approach the optimal solution over time. If it can find better successive state, it takes successive state. Or it accepts worse state with a certain probability, with the probability of jumping to bad move decreases with temperature. This is less extreme than taking randomized hill climbing each time but still has the ability of escaping from possible trap of local maximum/minimum or plateau. The pseudo code of simulated annealing is listed below.

```
current is the initial state
schedule is a mapping from search time to temperature

for each search time step t
    T = schedule[t]
    if T = 0 RETURN current
    next is a random successor of current
    ΔE = VALUE[next] - VALUE[current]
    if ΔE > 0 then current is next
    else current is next with probability e ^ (ΔE/T)
```
When applying simulated annealing, some related functions has to be defined first, and are subject to changes in order to observe the impact on the behavior (speed of convergence and tolerance on accuracy) of problem solving process:

1. **Successive States:**
   One of the criteria in selecting algorithms for successive states is that it is crucial to make sure that selection of successive state algorithm includes does include optimal route as one of its descendent states. In this implementation, successive states are generated from random swapping between two cities in the route. This selection is brute force, and choice of other successive state algorithm can be derived from graph theorem. However, the randomness of successive states can be used to test the robustness of the simulated annealing algorithm.

2. **Temperature Mapping Function:**
   Temperature mapping function decide how fast it would converge to hill climbing. When temperature tends to 0, simulated annealing simply becomes hill climbing. The same as real annealing, the temperature should only be going down in order to stabilize, if the temperature is not strictly decreasing against time, the vibration to worse route will intensify over time, leading to unstable conditions. In this implementation, there are four mapping functions to choose from, which are linear decreasing, reciprocal, time and time.

3. **End Condition:**
   After close examination of the pseudo code of the algorithm in the textbook and search on the web, there is no defined end condition for simulated annealing. Some of the documentation suggests the use of a certain time window, which makes it the algorithm that gives the best estimate on given time. The other way maybe to use minimal spanning tree to admissibly estimate the lower bound of cost function, and setup a threshold that is some magnitude multiples of this lower bound, but the multiples needed to create the threshold that is close enough to optimal yet not unreachable becomes major issue. In this implementation, end condition is set based on a parameter called “stable_count”, it counts the number of iterations that does not alter the route since last change. The threshold is always setup correlated to number of cities. If the stable_count reaches this threshold, it is declared that the simulator finds an goal route (either optimal or suboptimal).

4. **Randomized seed**
   Randomized seeds are used in creating initial city distribution. They are all allocated on Euclidean plane of 500 X 500 space with integer coordinates. The randomized seed can be specified by user, hence providing the option for simulating in repetitive environment. It is also used in creation of initial route to start with and the choice of swapping between cities.
Empirical Results

To make TSP solver, please type “gcc –o tsp tsp.c –lm”. To run this TSP solver, please type “./tsp [debug-level] [number-of-city] [seed for city creation] [algorithm]”.

Where higher debug-level gives more detailed information printout to show the progress of the solver. Number of the city can not be larger than 250, and is default at 100 cities. The algorithm option is default as Simulated Annealing, but user can specify 1 for one-shot hill climbing, 2 for randomized hill climbing, and 3 for stimulated annealing.

The empirical results conducted using TSP solver is shown below, all the measurements are done with city-seed equals 50 (which means the same set of data as long as number of cities stays the same), the threshold is set such that stable_count must be greater than double of number of cities and with reciprocal temperature mapping function for simulated annealing:

<table>
<thead>
<tr>
<th>Path length/iteration/time</th>
<th>25 cities</th>
<th>100 cities</th>
<th>200 cities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill climbing</td>
<td>2060.58/2371/.022</td>
<td>4330.86/33801/.613</td>
<td>5950.76/153816/283.72</td>
</tr>
<tr>
<td>Randomized hill</td>
<td>2067.24/1767/.021</td>
<td>4200.19/38760/36.6</td>
<td>5876.00/175360/559.56</td>
</tr>
<tr>
<td>Annealing</td>
<td>2103.08/16289/.14 4</td>
<td>4195.64/105093/22.0 0</td>
<td>5843.07/1198000/129.7 1</td>
</tr>
</tbody>
</table>

All the measurement above has been tested 3 times and taken the average from them. All the time measurement is in seconds, iteration for randomized hill climbing is the average value, and therefore total iteration should multiply by half of the number of the cities.

Here is some result of applying measurement to 100-city scenario with different temperature mapping functions. It still use the same set of 100 cities as in previous experiment, and result is shown as following.

<table>
<thead>
<tr>
<th>Path length/iterations/time</th>
<th>Reciprocal fet. (1/iteration)</th>
<th>Inverse square (1/iteration^2)</th>
<th>Inverse cube (1/iteration^3)</th>
<th>Linear Max(0,10^7-iteration)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 cities</td>
<td>4195.64/1050937/22.0</td>
<td>4313.19/85208/3.99</td>
<td>4098.70/126496/2.69</td>
<td>Can’t finish in 1 hr</td>
</tr>
</tbody>
</table>
The measurement set to test the threshold value of stable_count that is cubic to the number of cities never converges even after 2 hours. Therefore it is assume that threshold is too high to have practical use.

**Discussion and Conclusion**

1. **Robustness**: Stimulated Annealing is a fairly robust algorithm, given any random set of scattered cities, any randomly chosen initial route, the produced route always has length that is within 2.5% of deviation from mean value. Where hill climbing has the highest, and randomized hill climbing is in between these two algorithms.

2. **Fast convergence**: Even with the number of cities as high as 400, stimulated annealing is still able to converge within half an hour. As for the 200-city scenarios, it typically finishes within 3 minutes. Comparison made to randomized hill climbing, in order to gain comparative result of the same low-deviation routes, the convergence time is different by a factor of 1.5, and growing when the number of cities increases. This fact may have to do with the number of rounds that randomized hill climbing is ran, which is set to half of the number of the cities. However, this is the set that is found to conform to the performance of stimulated annealing.

3. **Threshold criteria**: There is no mentioning in the textbook about how stimulated annealing is going to terminate. It is decided that stable_count seems to be a good measurement, because it shows how many attempts that the simulator has made to find a better (or worse but acceptable) successive state. If it can be shown that it can not switch to another successive state within iterations equal to square of the number of cities, then the probability of it getting close to a local optimal/sub-optimal route is high. The result of simulation supports this decision, since comparison to the result given by randomized hill climbing is within 1% of difference. A better prove can be observed during the process of taking cubic threshold, once the stable count pass square threshold, the variation of route length is generally sub-decimal, hence proves the solution given is reasonably close to optimal solution.

Other implementation from the web either takes time limit, number of iteration, or suggestion a scale factor of admissible estimation using algorithms such as minimum spanning tree to be used as threshold. They have good reasoning behind these suggestions, but stable count seems to be working fine too.

4. **Different temperature mapping**: In this project I experiment with linear, reciprocal, square inverse, cubic inverse temperature mapping function. It turns out constant and linear mapping function cause too much vibration between current and
possible next state once the algorithm converges close to solution, because $\Delta E$ is so small that probability of switching over becomes so close to one, while temperature is either not growing or growing too slowly for the program to be stabilized. Reciprocal and inverse square seems to be better choices, they allow some worse move at the beginning, but converges well toward the end. Inverse square seems to be converging faster in general. Inverse cube is considered to be too aggressive, it behaves very similar to hill climbing that it seems to depend more on the probability rather than actual convergence, and it converges almost as fast as hill climbing.

5. The benefit of using simulated annealing algorithm in random selecting updates seems not obvious, according to Brian T. Luke (btluke.netfirms.com/apsatsp.html) It seems that using some simpler algorithm such as greedy search (shortest path first) will also generate a result not too far off the result gained by simulated annealing, while the cost of getting that solution is definitely much lower than simulated annealing.

6. Cost of finding route is still growing exponentially: Unfortunately, it is still observed that even when using algorithm such as stimulated annealing, the time and iteration needed to converge still grows exponentially. It can be observed from the table shown above, when number of cities grow from 25 $\rightarrow$ 100, the time required climbed more than 2000 times, while number of iteration grows 6 times. When the number again increases to 200 cities, the time needed grows 6 times, and the iteration is now 11 times.
However, if considering exhaust search of 11 cities will take 40 million permutations, it can be sure that simulated annealing is definitely a feasible solver for TSP problem.