Group Project 2

TRAVLING SAILSMAN PROBLEM

PROJECT GROUP 5

Table of Contents

For this project we picked three different types of algorithm to explore the different ways to solve the problem. These were the Bellman-Held-Karp (dynamic) Algorithm a greedy algorithm, and an algorithm unlike anything we worked on in class the simulated annealing algorithm.

Bellman-Held-Karp Algorithm

Description:

This is the only Dynamic program I could find that was proposed as a solution to the traveling salesman problem. It was found independently by Bellman and by Held and Karp. This algorithm works on the premise that we don't know what to solve so it solves all sub problems. And all efficient sub problems will make an efficient main problem.

Example:

We have a 4 nodes it would calculate the best result between following the paths 2341, 21341, 3241,4231 where it will then recursively call its self and check all nodes minus the one it just check thus we get something like this

With four nodes to simplify

1,2,3,4

(1,2)[234],   (1,3)[324],   (1,4)[432]

(2,3)[34], (2,4)[34],| (3,1)[24], (3,4)[24],| (4,2)[32], (4,3)[32]

(34)[4]|(43)[3] |(24)[4]|(42)[2] |(23)[3]|(32)[2]

(41)   (31)   (41)   (21)   (31)   (21)

Resorce useage:

This program exponential which is much faster than logarithmic brute force method however it suffers from using allot of space.

While the wost case when processing it is   making this algorithum much better than the brute force method but still quite slow.

Psudo-code:

```
Findpath(int(* rawData)[len], int numNodes)
    int i, j
    for (J=0; j< numNodes;j++)
        for (i=0;i<numNodes;i++)
            data[i][j] = distance(rawData[1][j],rawData[2][j],rawData[1][i],rawData[2][i],)
    while true
        int refindpath(0, nodes[],numNodes-1, data, path[])

int refindpath(primary, nodes[],numNodes, data[][], path[])
int result[]
    int best[]
    int node
    int temp
    best = MAX VALUE;
    if numNodes>=1
        for (int i = 0;i<numNodes;i++)
            best[i] = data[primary][nodes[i]][refindpath(nodes[i], removenode(nodes[],i), data, path[])
        return pickBest(best)    // returns the best value it can find
    else
        retun 0
```

Conclusion:

Though an interesting algorithm this one is too slow to be used for the competition but it is guaranteed to give you the best answer.

Greedy algorithm

Description:

Greedy algorithm is to find the best solution on each step. For tsp problem, greedy can find a shortest tour form start city to visited all cities and back to start city without visited same city twice.

A greedy algorithm for the traveling salesman problem: Pick an arbitrary city and call it city 1. Find a city with the smallest distance from city 1, and call it city 2. Find a city in the rest of the n − 2 cities with the smallest distance from city 2 . . .

Output the tour: City1→City2→⋯→Cityn→City1.

Pseudo code:

```
def nearestneighbor(cities)
    start_vertex = vertices[0]
    unvisited_vertices = vertices[1:]
    tour = [start_vertex]
    while len(unvisited_vertices) > 0:
        nearest_neighbor = find_nearest_neighbor(tour[-1], unvisited_vertices)
        unvisited_vertices.remove(nearest_neighbor)
        tour.append(nearest_neighbor)
   return tour
```

Simulated annealing algorithm:

Description:

Simulated annealing is a randomized algorithm which find the optimal answer from a set of solutions in the required time. When simulated annealing algorithm is used to solve TSP, it is different from the greedy algorithm which always take the minimum path and set it to optimal. The annealing algorithm may choose the “worse” solution(longer than the current shortest path) rather than always find the shortest path, since if we always choose the minimum path, we may get the partial optimal solution rather than the best one eventually. The rule of choosing “worse” solution is called annealing.

Here is the annealing rule:

Let T_min be the minimum temperature, T_int be the initial temperature, and k be the coefficient of temperature decrease. In the theory of thermodynamics, P(dE) = exp(dE/kT), in which dE is the different energy between the current and dropped temperature. From the formula, it is not hard to find that when current temperature is high, the possibility of dropping temperature in energy size of dE (called “P(dE)”) is bigger than the lower current temperature. Since this is annealing, dE < 0, dE/kT < 0. Thus, 0 < P(dE) < 1.

In TSP, when a length of path incoming, the algorithm always check the temperature, if it is bigger than T_min, and this path is longer than current optimal path, then using the P(dE) formula to find the possibility of using this path or not.

Here is the pseudo code of Simulated annealing:

```
Find a new distance x1
Old distance is x0
If x1 < x0
    Then x0 = x1
else if T > T_min and P(dE) in range(0,1)
    Then x0 = x1
T *= k
```

Other pseudo code of the simulated annealing algorithm:

1) Using the calculated distance matrix of all cities to calculate every solution path.

```
Length_calculate(dist, path)
    Len = 0
    Path is the sequence of the current solution of path
    Len += all length of edges in path
    Len += the length of last vertex to the first vertex in path
```

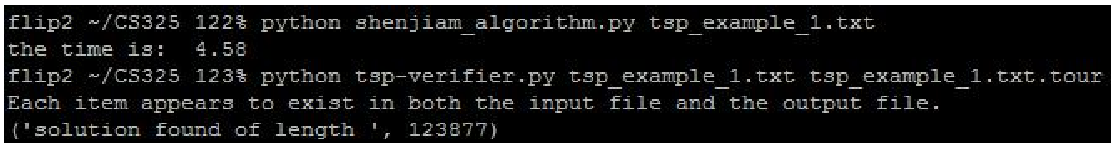
2) Using two kinds of idea to get the new path. One way is to change the order between two random node in old path, the other is randomly find three node x,y,z and change edge x-z to x-y and y-z. It can also add more method of changing the path such as reverse the old path or move all node to the left by one. More methods can make the algorithm faster and it is better to use ratio to control the possibility of using these methods.

```
Find_new_path_1(old_path)
    Randomly get two nodes x1 and x2.
    If x1 != x2
        Then change the order of x1 and x2, new_path[x1] = old_path[x2]
Find_new_path_2(old_path)
    Randomly get three nodes x1, x2 and x3
    If x1 != x2 != x3
        Then let E(a-c) = E(a - b) + E(b - c)
```

Experiment:(This is not the algorithm we submitted!)

Just for the experiment, I implement this algorithm to compare with two others.(This is not our final algorithm since it takes long time to solve case 3)

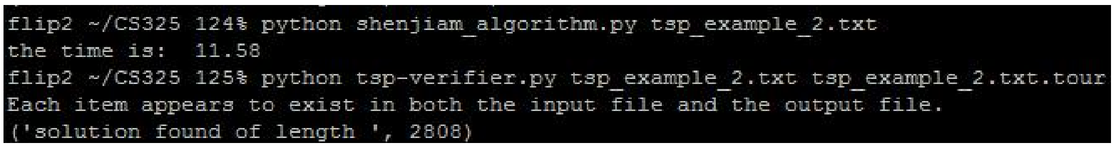
Here is the result of testcast1 and testcase2:



```
flip2 ~/CS325 122% python shenjiam_algorithm.py tsp_example_1.txt
the time is:  4.58
flip2 ~/CS325 123% python tsp-verifier.py tsp_example_1.txt tsp_example_1.txt.tour
Each item appears to exist in both the input file and the output file.
('solution found of length ', 123877)
```

The time of the first case is 4.58, and the path is 123877.

The ratio is 123877/108159 = 1.145



```
flip2 ~/CS325 124% python shenjiam_algorithm.py tsp_example_2.txt
the time is:  11.58
flip2 ~/CS325 125% python tsp-verifier.py tsp_example_2.txt tsp_example_2.txt.tour
Each item appears to exist in both the input file and the output file.
('solution found of length ', 2808)
```

The time of the second case is 11.58, and the path is 2808.

The ratio is 2808/2579 = 1.0888

There is no test case for case3 since even in the flip, the speed is too slow to get the accurate answer.

Conclusion:

Overall, in the small case of cities, this algorithm is better than greedy algorithm since it can get more accurate solution. However, this algorithm is based on exhaustive method, even it is faster than the exhaustive(O(n!)) the speed of this algorithm still cannot faster than greedy.

Overall conclusion and final result

We implement greedy algorithm for our final answer and the test case for all example and test inputs below:

Best tours for the three example instances and the time it took to obtain these tours.

	Tour	Time(sec)
Example 1	130921	0.198181
Example 2	2975	8.574694
Example 3	1964948	84.758068

Best solutions for the competition test instances. Time limit 3 minutes and unlimited time.

	Tour	Time(sec)
Test-input-1	5911	0.125849
Test-input-2	8011	0.754211
Test-input-3	14826	11.497598
Test-input-4	19711	58.614923
Test-input-5	28685	0.792768
Test-input-6	40933	3.113752
Test-input-7	63780	19.571800

As a result over all we have found that the best ways to solve this problem are with algorithms not attempting to find the true perfect answer. As such algorithms such as greedy or genetic run much faster than more precise ones such as Held-Karp or simulated annealing

Reference:

Simulated annealing. (2017, August 15). Retrieved August 17, 2017, from https://en.wikipedia.org/wiki/Simulated_annealing

"Held–Karp algorithm." Wikipedia, Wikimedia Foundation, 15 May 2017, en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm. Accessed 19 Aug. 2017.