Project Final Report

Group 150: Zhiyi Li, Yao Shao, Huben Liu

December 2019

1 Algorithm Main Idea

The general idea of the algorithm is to go from the special case to the general case. Since one of the extreme cases is to send all TAs home by car, we first find a route that going through all TAs homes.

Before we do this, we first define a function dfs to get the length of all possible paths between two given vertices by floyed algorithm, storing them in an array. By calling dfs function, we have access to all length of the possible pairs of vertices.

The problem to find a route going through all TAs homes is similar to Traveling Salesman Problem. But the setting of TSP problem is on the complete graph, however our graph input may not be complete. But the graph is connected. So by the floyed algorithm stated above, we have all paths and its length between every pair of vertices, then we can treat the graph as a complete graph. In addition to the triangle inequality in the problem setting, we can solve this problem as TSP.

There are several methods for tsp and MST approximation is a very common one, but its not so efficient compared to some other approximation algorithms. Simulated annealing can get a good approximation effect in most cases when solving the tsp problem. We finally use the simulated annealing method to solve the tsp problem.

In general, if we use the exactly greedy line search, we are going to find the local minimum of the object function, but it may not be the global optimal. By simulated annealing, we are probably going to hit the global minimum. The general idea is in every iteration, say iteration $i+1$ for example, we generate a new route going through all the objectives (TAs homes), we call it $R(i+1)$. Return the cost of $R(i+1)$, call it $c(R(i+1))$. If $c(R(i+1)) < c(R(i))$, we accept the new path. Else, we accept the new path with probability by simulated annealing. The probability function we use is

$$e^{\frac{c(P(i))-c(P(i+1))}{T}}$$

Then cool down (decrease T) and go to another iteration until we meet the final requirement. In our algorithm, the requirement is $T < 10^{-16}$ And we run simulated annealing 17 times to pick a optimal route going through all TAs home after this step. Assume the cost of this route is c_0

Now we are going to further optimize the path. We choose one home v_i from the car route with all other homes fixed and find all possible alternate drop off place for the TA whose home is v_i. Since we have the primal route going through all TAs homes, we assume the one house before v_i is v_{i-1}, the one house after v_i is v_{i+1}. So we fix v_{i-1} and v_{i+1}, check all vertices on any possible paths going from v_{i-1} to v_{i+1} if it is a better dropping off place for the TA_i (we denote the TA whose home is v_i the TA_i). Because the car cost 2/3 and people cost 1 on every path, all other TAs will not have alternate dropping off point except for their home. After this step, we assign an optimal drop off place for TA_i, call that v_i^*. We replace old path going through v_{i-1}-v_i-v_{i+1} by the

new path v_{i-1}-v_i^*-v_{i+1} we have just find. Update c_0 by the lower cost of this new path. Then check if v_{i+1} can be replaced by a better point, say v_{i+1}^* for TA_{i+1} to get off the car. Then Update the path. Continue the iteration until all homes checked. Return the final output route and the cost.

It is worth mentioning that since our approach to get the primal case is somehow random, though we iterate 17 times at one run. So the more we run the algorithm, the more chance we will have getting better solution.

2 Pseudocode of the Most Important Part

In this part, we give simplified pseudocode of the most important part of our algorithm to make our idea clear. The actual algorithm is far more longer and includes many processes such as read in, initialize, generate,etc.

function dfs(v_i,v_j) //Get all possible path between v_i and v_j

function generate() //Generate paths going through all TAs homes

function SA(): //Simulated annealing
  initial T=1
  while itertimes:=200000,....,2,1:
    generate a new path, length l_new
    df=l_new-l_0
    if $df < 0$: pick the new path. l_0=l_new
    elif $e^{-df/T}$ >random number $\in$ [0,1], pick the new path. l_0=l_new
    T=T * 0.999
    if $T < 10^{-16}$, break

Main function:

//generate the lowest route through all TAs home
for i:=1,2,...,17
  SA() and keep track of the route R with the lowest cost c_0.

//update the route
  for every $V_i \in$ R
    for $path_i \in$ {every path between v_{i-1} and v_{i+1} }:
      for $v' \in$ { $v|v \in path_i$ }:
      newcost = c_0 - carcost*(length(v_{i-1},v_{i+1}) in R) + carcost*(length(v_{i-1},v',v_{i+1}) in $path_i$) + mancost* shortest path(v', v_i)
        if $newcost < c_0$, update R, replace path (v_{i-1},v_{i+1}) in R with $path_i$. c_0=newcost, v_i=v'
//Return the final solution

3 Greedy Approach

We can also use the greedy approach to get a TSP solution in part one by going to the nearest TAs home every time. This will make the algorithm easier but in general SA gives solution with lower cost. But it is still worth trying just because it may give a better solution in case.

4 Computing Resources

Group members own laptops.
CPU:
Intel(R) Core(TM) i5-6200U CPU @ 2.3GHz,
Intel(R) Core(TM) i5-8250U CPU @ 1.6GHz,

Total Run time:
Run algorithm around 15 times, 30 hours.
(With our code implemented in C++)